8 Размещение виджетов на фрейме

Эта глава включает

- Создание фреймов и применение стилей
- Работа с фреймами и полосами прокрутки
- Создание альтернативных типов фреймов
- Создание и манипулирование окнами-разделителями

Всё взаимодействие пользователя с Вашей wxPython-программой происходит в виджете-контейнере, который обычно принято называть окном. В терминологии wxPython такой контейнер называется фреймом. В этой главе мы обсудим несколько различных фреймовых стилей wxPython. Основной класс `wx.Frame` содержит несколько различных стилевых флагов фрейма, влияющих на его внешний вид. Кроме того, wxPython предлагает минифреймы (miniframe) и фреймы, которые реализуют многодокументный интерфейс (MDI). При помощи разделительных полос (splitter bar) фреймы могут быть разделены на части, а при помощи полос прокрутки (scrollbar) они могут включать панели, превосходящие по размеру сам фрейм.

8.1 Жизненный цикл фрейма

Мы начинаем с обсуждения наиболее общих элементов фреймов: с их создания и размещения. Создание фрейма предполагает знание обо всех тех элементах стиля, которые могут быть использованы; размещение фреймов возможно более сложная задача, чем Вы можете предполагать с самого начала.

8.1.1 Как создать фрейм?

В этой книге мы уже рассмотрели множество примеров создания фрейма, но, все же, рискуя повторяться, мы выполним обзор базовых принципов создания фреймов.

Создание простого фрейма

Фреймы являются экземплярами класса `wx.Frame`. Листинг 8.1 показывает очень простой пример создания фрейма.

Листинг 8.1 Базовый код создания `wx.Frame`

```
import wx

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = wx.Frame(None, -1, "A Frame",
        style=wx.DEFAULT_FRAME_STYLE,
        size=(200, 100))
    frame.Show()
    app.MainLoop()
```

Здесь создается фрейм с заголовком "A Frame" и размером 200 на 100 пикселей. Использованный в листинге 8.1 стиль фрейма по умолчанию обеспечивает его стандартное оформление с кнопками закрытия, минимизации и максимизации. Рисунок 8.1 показывает результат выполнения данного кода.

Рисунок 8.1 Простой фрейм

Конструктор wx.Frame подобен другим конструкторам виджетов, которые мы видели в главе 7:

```
wx.Frame(parent, id=-1, title="", pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE,
        name="frame")
```

`wx.Frame` имеет дюжину характерных ему стилевых флагов, которые мы рассмотрим в следующем разделе. Стиль по умолчанию обеспечивает Вас заголовком, кнопками минимизации и максимизации, выпадающим системным меню и утолщенной рамкой с изменяемым размером. Он подходит для большинства необходимых Вам стандартных окон приложения.

У `wx.Frame` нет сопоставленных с ним типов событий, за исключением тех событий, которые относятся к любому другому виджету. Тем не менее, раз `wx.Frame` является именно тем элементом на Вашем экране, который пользователь наиболее вероятно будет закрывать, у Вас обычно будет необходимость в определении обработчика для события закрытия, обеспечивающего правильное управление суб-окнами (подчиненными окнами) фрейма и данными.

Создание фреймового подкласса

У Вас нечасто будет необходимость создавать экземпляры `wx.Frame` непосредственно. Почти в каждом втором примере этой книги мы видим, что в типичном приложении wxPython создаются подклассы `wx.Frame` и экземпляры этих подклассов. Всё дело в уникальном статусе `wx.Frame` – сам по себе он определяет весьма несущественное поведение, а вот его подкласс с уникальным инициализатором представляет наиболее разумное место для размещения информации о компоновке и поведении Вашего фрейма. При выполнении манипуляций с Вашими специализированными формами и данными вполне возможно обходиться без создания подклассов, но за исключением простейшего приложения это трудно преодолевать. Листинг 8.2 показывает пример подкласса `wx.Frame`.

Листинг 8.2 Простой подкласс фрейма

```
import wx

class SubclassFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Frame Subclass',
                size=(300, 100))
        panel = wx.Panel(self, -1)
        button = wx.Button(panel, -1, "Close Me", pos=(15, 15))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)

    def OnCloseMe(self, event):
        self.Close(True)
```

```
    def OnCloseWindow(self, event):
        self.Destroy()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    SubclassFrame().Show()
    app.MainLoop()
```

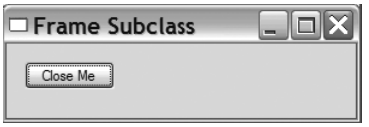

Рисунок 8.2 Простой фрейм в качестве подкласса

Результирующий фрейм такой же, как и на рисунке 8.2. Мы видели эту основную структуру во многих других примерах, поэтому давайте обсудим некоторые аспекты данного кода, характерные именно фреймам. Вызов метода `wx.Frame.__init__` имеет тот же формат, что и конструктор `wx.Frame`. Конструктор для подкласса не имеет параметров, что позволяет Вам как программисту определить параметры, передаваемые родительскому классу, и препятствует повторному определению одних и тех же параметров.

В листинг 8.2 заслуживает внимания также и то, что встроенные суб-виджеты фрейма сами размещены внутри панели. Панель является экземпляром класса `wx.Panel` и это простой контейнер для других виджетов с ограниченной собственной функциональностью. В качестве высокоуровневых суб-виджетов Вашего фрейма Вам необходимо практически всегда использовать `wx.Panel`. Прежде всего, такой дополнительный уровень делает код лучше в плане повторного использования, так как одни и те же панель и компоновка могут быть использованы в разных фреймах. Использование `wx.Panel` обеспечивает Вам в пределах фрейма некоторую диалоговую функциональность. Такая функциональность проявляет себя несколькими путями. Во-первых, в операционной системе MS Windows экземпляры `wx.Panel` имеют по умолчанию другой цвет фона – белый вместо серого. Во-вторых, панели могут иметь элемент по умолчанию, который автоматически активизируется при нажатии клавиши Enter, и к тому же они практически так же, как это делает диалог, реагируют на клавиатурные события при табуляции между своими элементами или выборе элемента по умолчанию.

8.1.2 *Какие существуют другие стили фрейма?*

Класс `wx.Frame` имеет большое число возможных стилевых флагов. Обычно всё, что Вам нужно – это стиль по умолчанию, но есть и несколько полезных вариантов. Первый обсуждаемый нами набор стилевых флагов управляет общей формой и размером фрейма. Хотя нет строгих требований, но эти флаги должны взаимно исключаться, т. е. данный фрейм должен использовать только один из этих флагов. Использование стилевого флага этой группы не предполагает существование каких-либо флагов оформления, описанных в других таблицах этого раздела; Вам необходимо комбинировать флаг формы вместе с другими

необходимыми флагами оформления. Флаги формы и размера описывает таблица 8.1.

Таблица 8.1 Стилевые флаги формы и размера фрейма

Флаг стиля	Описание
`wx.FRAME_NO_TASKBAR`	Вполне нормальный фрейм, за исключением одной вещи: он не отображается в панели задач (taskbar) под Windows и другими поддерживающими эту возможность системами. При минимизации фрейм сворачивается в иконку на рабочий стол, а не в панель задач. (Таким путем фреймы вели себя до версии Windows 95).
`wx.FRAME_SHAPED`	Непрямоугольный фрейм. Точную форму фрейма устанавливает метода `SetShape()`. Формованные окна будут обсуждаться в этой главе позже.
`wx.FRAME_TOOL_WINDOW`	Такой фрейм имеет меньшую, чем в нормальном состоянии полосу заголовка, обычно используется для вспомогательных фреймов, которые содержат ряд инструментальных кнопок. Под операционными системами Windows, инструментальное окно в панели задач не отображается.
`wx.ICONIZE`	Это окно в своём исходном положении будет показано минимизированным. Данный стиль имеет эффект только в операционных системах Windows.
`wx.MAXIMIZE`	Это окно в своём исходном положении будет показано развернутым (полноэкранным). Данный стиль имеет эффект только в операционных системах Windows.
`wx.MINIMIZE`	Тоже, что и `wx.ICONIZE`

Из этой группы флагов более всего нуждается в скриншоте стиль `wx.FRAME_TOOL_WINDOW`. Рисунок 8.3 показывает небольшой пример использования флага `wx.FRAME_TOOL_WINDOW` вместе с флагами `wx.CAPTION` и `wx.SYSTEM_MENU`. Если по этой картинке Вы не можете представить масштаб, мы Вам гарантируем, что полоса заголовка этого инструментального фрейма *уже*, чем у других фреймовых стилей.

Рисунок 8.3 Образец стиля инструментального окна

Есть два взаимно исключающих стиля, которые управляют тем, остается ли фрейм сверху других фреймов, даже когда получают фокус другие фреймы. Это может быть полезно для небольших диалогов, которым нет необходимости оставаться на виду длительное время. Эти стили описывает таблица 8.2.

Наконец, имеется несколько элементов оформления, которые Вы можете разместить в Вашем окне. Вы должны их явно указать, иначе, если Вы оставите заданный по умолчанию стиль, они не установятся автоматически, и легко

получится окно, которое невозможно закрыть или переместить. Список этих стилей оформления приводит таблица 8.3.

Таблица 8.2 Стили плавающего поведением фрейма

Флаг стиля	Описание
`wx.FRAME_FLOAT_ON_PARENT`	Фрейм размещается поверх своего родителя, и только его. (Очевидно, что для использования этого стиля, фрейму нужно иметь родителя). Другие фреймы будут перекрывать этот фрейм.
`wx.STAY_ON_TOP`	Фрейм всегда будет находиться сверху любого другого фрейма в системе. (Если Вы определили с этим флагом более одного фрейма, такие фреймы относительно друг друга будут перекрываться нормально, но все же будут находиться сверху всех других фреймов системы.)

Стиль по умолчанию `wx.DEFAULT_FRAME_STYLE` эквивалентен набору флагов `wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.CLOSE_BOX | wx.RESIZE_BORDER | wx.SYSTEM_MENU | wx.CAPTION`. Этот стиль создает стандартное окно, для которого применимы операции изменения размеров, минимизации, максимизации и закрытия. Хорошая идея состоит в том, чтобы формирование других стилей, начиналось со стиля по умолчанию – это гарантирует, что Вы получите правильный вид оформления. Например, для создания инструментального фрейма Вы можете использовать следующий стиль: `wx.DEFAULT_FRAME_STYLE | wx.FRAME_TOOL_WINDOW`. Помните, что для удаления флага из маски можно использовать оператор `^`.

Таблица 8.3 Стили оформления окна

Флаг стиля	Описание
`wx.CAPTION`	Выводит окно с полосой заголовка. Вы должны включить этот стиль, чтобы разрешить другие элементы, которые устанавливаются здесь по традиции (кнопка минимизации и максимизации, системное меню и контекстная помощь).
`wx.FRAME_EX_CONTEXTHELP`	Этот стиль относится к операционным системам Windows и устанавливает в правом углу полосы заголовка иконку «Помощь» в виде знака вопроса. Этот стиль взаимно исключает флаги `wx.MAXIMIZE_BOX` и `WX.MINIMIZE_BOX`. Он является расширенным стилем и должен быть добавлен при помощи двухшагового процесса создания, который описывается позже.
`wx.FRAME_EX_METAL`	В Mac OS X фреймы с этим стилем будут иметь вид отшлифованного металла. Это дополнительный стиль, который должен устанавливаться методом `SetExtraStyle`.

wx.MAXIMIZE_BOX	Устанавливает кнопку максимизации в стандартном месте полосы заголовка.
wx.MINIMIZE_BOX	Устанавливает кнопку минимизации в стандартном месте полосы заголовка.
wx.CLOSE_BOX	Устанавливает кнопку закрытия в стандартном месте полосы заголовка.
wx.RESIZE_BORDER	Устанавливает фрейму нормальную границу с манипуляторами изменения размеров.
wx.SIMPLE_BORDER	Устанавливает фрейму миниатюрную границу без возможности изменения размеров и без элементов оформления. Этот стиль взаимно исключает все другие стили оформления.
wx.SYSTEM_MENU	Устанавливает в полосе заголовка системное меню. Точное содержание системного меню соответствует другим выбранным стилям оформления. Например, у Вас будет пункт "Свернуть" только, если указан флаг wx.MINIMIZE_BOX.

8.1.3 *Как создать фрейм с дополнительной стилевой информацией?*

Рисунок 8.4 Фрейм с дополнительной контекстной помощью

Стиль wx.FRAME_EX_CONTEXTHELP является расширенным стилем, который означает, что значение этого флага слишком велико и не может устанавливаться при помощи обычного конструктора (из-за специфических ограничений соответствующего типа переменной C++). Обычно Вы можете установить дополнительные стили после создания виджета методом SetExtraStyle, но некоторые стили, например, wx.FRAME_EX_CONTEXTHELP, должны быть установлены прежде, чем будет создан соответствующий объект интерфейса пользователя. В wxPython это выполняется слегка неуклюжим способом, известным под именем *«двухшаговая конструкция»* (*twostep construction*). Как показано на рисунке 8.4, после использования этой конструкции создается фрейм, имеющий в полосе заголовка иконку в виде знака вопроса.

Значение флага должно быть установлено с помощью метода SetExtraStyle(). Иногда, дополнительная стилевая информация должна быть установлена до создания экземпляра фрейма, что приводит к философскому вопросу о том, можете ли Вы вызвать метод экземпляра, которого еще не существует. В следующих разделах мы покажем два механизма выполнения этой операции, при этом второй из них будет общей абстракцией первого.

Добавление дополнительной стилевой информаци

В wxPython дополнительная стилевая информация добавляется перед созданием фрейма при помощи специального класса `wx.PreFrame` (пред-фрейм), который является частичного инициированным типом фрейма. Вы можете установить в пред-фрейм дополнительный стилевой бит, а затем, используя данный пред-фрейм, создать фактический экземпляр фрейма. Листинг 8.3 показывает, каким образом в конструкторе подкласса выполняется «двухшаговая конструкция». Обратите внимание на то, что в wxPython реально это трех-шаговый процесс (в инструментальном комплекте C++ wxWidgets это двухшаговый процесс, отсюда такое название).

Листинг 8.3 Двухфазное создание окна

```
import wx

class HelpFrame(wx.Frame):

    def __init__(self):
        pre = wx.PreFrame()
        pre.SetExtraStyle(wx.FRAME_EX_CONTEXTHELP)
        pre.Create(None, -1, "Help Context", size=(300, 100),
                style=wx.DEFAULT_FRAME_STYLE ^
                (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        self.PostCreate(pre)

if __name__ == '__main__':
    app = wx.PySimpleApp()
    HelpFrame().Show()
    app.MainLoop()
```

Предварительно сконструированный объект

Этот вызов создает фрейм

Передача соответствующих C++ указателей

1. Создание экземпляра `wx.PreFrame()` (аналог для диалогов - `wx.PreDialog()`, другие виджеты wxWidgets имеют свои собственные предклассы). После этого вызова Вы можете выполнять любые другие необходимые Вам инициализационные мероприятия.
2. Вызов метода `Create()` имеет тот же вид, что и конструктор wxPython.
3. Эта строка характерна для wxPython и она не выполняется на C++. Метод `PostCreate` выполняет некоторые внутренние служебные мероприятия, которые представляют Ваш wxPython-экземпляр в виде интерфейса к созданному Вами на первом шаге объекту C++.

Добавление дополнительной стилевой информации с обобщением

Приведенный выше алгоритм несколько неудобен, но для более простого управления его можно рефакторизовать. На первом шаге необходимо создать общую вспомогательную функцию, которая может управлять любым

двухшаговым процессом создания. Листинг 8.4 содержит пример использования рефлективной (возвратной) способности Python для вызова произвольных функций, передаваемых в качестве переменных. Этот пример целесообразно использовать для вызова в методе `__init__` в течение инициализации Python нового фрейма.

Листинг 8.4 Общая функция двухшагового создания

```
def twoStepCreate(instance, preClass, preInitFunc, *args,
        **kwargs):
    pre = preClass()
    preInitFunc(pre)
    pre.Create(*args, **kwargs)
    instance.PostCreate(pre)
```

Функция в листинге 8.4 принимает три обязательных параметра. Параметр `instance` – это фактический создаваемый экземпляр. Параметр `preClass` это классовый объект для временного предкласса, в случае фреймов – это `wx.PreFrame`. Параметр `preInitFunc` это объект-функция, которая обычно будет обратным вызовом (callback) в методе инициализации экземпляра. Вслед за этим может быть добавлено произвольное количество других дополнительных параметров.

Первая строка этой функции, `pre = preClass()`, рефлективно инициализирует предварительно создаваемый объект, используя переданный в качестве параметра классовый объект. Следующая строка соответственно вызывает функцию обратного вызова, переданную в данном случае как `preInitFunc`, которая обычно устанавливает расширенный флаг стиля. Затем с использованием опциональных параметров вызывается метод `pre.Create()`. И, наконец, вызывается метод `PostCreate`, преобразующий внутренние значения из предварительных в экземплярные. В этой точке параметр `instance` создан полностью. Предполагая импортирование `twoStepCreate`, данная вспомогательная функция может быть использована, как показано в листинге 8.5.

Листинг 8.5 Еще одно двухшаговое создание с использованием общего метода

```
import wx

class HelpFrame(wx.Frame):

    def __init__(self, parent, ID, title,
                 pos=wx.DefaultPosition, size=(100,100),
                 style=wx.DEFAULT_DIALOG_STYLE):
        twoStepCreate(self, wx.PreFrame, self.preInit, parent,
                 id, title, pos, size, style)

    def preInit(self, pre):
        pre.SetExtraStyle(wx.FRAME_EX_CONTEXTHELP)
```

Класс `wx.PreFrame` и функция `self.preInit` переданы в основную функцию, и метод `preInit` определен в качестве обратного вызова.

8.1.4 Что происходит при закрытии фрейма?

Когда Вы закрываете фрейм, он, в конечном счете, пропадает. Кроме того случая, когда фрейму явно указано, чтобы он не закрывался. Другими словами, фрейм вовсе не такой прямолинейный. Замысел архитектуры закрытия виджета в wxPython состоит в том, чтобы дать закрывающему виджету достаточную возможность выполнить закрытие документов или освобождение любых внешних по отношению к wxPython ресурсов, которые могут удерживаться. Это - особенно приветствуется, если Вы удерживаете некоторый тип ценного внешнего ресурса, например, большую структуру данных или соединение с базой данных.

Общеизвестно, что поскольку C++ не управляет за Вас очисткой распределенной памяти, управление ресурсами в царстве C++ wxWidgets является одной из важнейших задач. В wxPython явная необходимость в многофазном процессе закрытия минимальна, однако это все еще может быть полезным, чтобы иметь дополнительный контроль над этим процессом. (Кстати, переход от слова *фрейм* к слову *виджет* в этом разделе выполнен осознанно – все выводы этого раздела относятся ко всем виджетам верхнего уровня, таким как фреймы или диалоги).

Когда пользователь инициирует процесс закрытия

Процесс закрытия в общем случае инициируется действием пользователя, например, щелчком на кнопке закрытия или выбором пункта *Закрыть* системного меню, а также когда приложение вызывает метод фрейма `Close` в ответ на некоторое другое событие. Когда это происходит, ядро wxPython вырабатывает событие `EVT_CLOSE`. Подобно любому другому событию wxPython, Вы можете связать с этим событием обработчик, вызываемый в ответ на `EVT_CLOSE`.

Если Вы не объявите собственный обработчик события, будет выполнена обработка по умолчанию. Такая встроенная обработка отличается для фреймов и диалогов.

- По умолчанию, фреймовый обработчик вызывает метод `Destroy()` и удаляет фрейм и все его виджеты.
- По умолчанию, обработчик закрытия для диалогов не уничтожает диалог – он просто имитирует нажатие кнопки отмены и прячет диалог. Объект диалога остается существовать в памяти, и при желании приложение может выбрать значения из виджетов с введенными данными. Когда приложение завершается как диалог, оно должно вызвать метод диалога `Destroy()`.

Если Вы пишете собственный обработчик закрытия, Вы можете его использовать для закрывания или удаления любых внешних ресурсов, но если Вы

решаете удалить фрейм, то ответственность за явный вызов метода `Destroy()` лежит на Вас. Хотя `Destroy()` часто вызывается из `Close()`, простой вызов метода `Close()` еще не гарантирует уничтожение фрейма. Вполне логично, что при определенных обстоятельствах будет решено не уничтожать фрейм, например, когда пользователь отменяет закрытие. Тем не менее, если Вы решаетесь его закрыть, все еще нужно иметь способ уничтожить фрейм. Если же Вы решаете не уничтожать окно, в этом случае лучше всего вызывать метод события закрытия `wx.CloseEvent.Veto()`, как сигнал для любой заинтересованной стороны о том, что фрейм отклонил приглашение своего закрытия.

Если Вы решите закрыть Ваш фрейм где-нибудь в пределах программы, не в обработчике закрытия, а например, из другого события пользователя, подобного пункту меню, рекомендуется выполнять вызов фреймового метода `Close()`. Это начинает описанный выше процесс точно тем же путём, каким выполняется системное закрытое. Для того чтобы убедиться в том, что фрейм действительно удален, можете непосредственно вызвать метод `Destroy()`, однако, это может привести к тому, что управляемые фреймом ресурсы или данные не будут освобождены или сохранены.

Когда процесс закрытия инициируется системой

Если событие закрытия инициируется непосредственно самой системой из-за завершения её работы (system shutdown) или чего-либо подобного, в этом случае имеется другое место, где Вы можете управлять этим событием. Класс `wx.App` получает событие `EVT_QUERY_END_SESSION`, которое позволяет Вам при желании налагать запрет на завершение приложения, сопровождаемое событием `EVT_END_SESSION`, если все выполняющиеся приложения одобряют такое выключение системы или GUI-среды. Реакция в случае наложения Вами запрета зависит от конкретной системы.

В конечном счете, стоит отметить, что вызов метода виджета `Destroy()` еще не означает, что виджет будет немедленно уничтожен. Уничтожение фактически будет выполнено, когда очередь событий станет пустой – т.е. после обработки любых событий, находящихся в ожидании на момент вызова `Destroy()`. Это предотвращает определенные проблемы, которые могут произойти, если события обрабатываются для виджетов, прекративших своё существование.

В следующих двух разделах мы переключимся от жизненного цикла фрейма к обсуждению некоторых вещей, которые Вы можете сделать с фреймом при его активности.

8.2 Использование фреймов

Фреймы содержат много методов и свойств. Наиболее важные из них – методы для поиска внутри фрейма произвольных виджетов, а также те, которые используются для прокрутки содержимого фрейма. В этом разделе мы обсудим их выполнение.

8.2.1 Методы и свойства класса wx.Frame

Таблицы этого раздела содержат наиболее важные свойства `wx.Frame` и его родительского класса `wx.Window`. Многие из этих свойств и методов рассматриваются в других местах книги более детально. Таблица 8.4 содержит некоторые общедоступные (public) читаемые и модифицируемые свойства `wx.Frame`.

Таблица 8.4 Общедоступные свойства `wx.Frame`

Свойство	Описание
`GetBackgroundColor()` `SetBackgroundColor(wx.Color)`	Цвет фона фрейма устанавливает цвет любой не занятой дочерними виджетами части фрейма. Вы можете передать в метод-установщик значение `wx.Color` или строку с именем цвета. Любая строка, передаваемая в метод wxPython в качестве цвета, будет интерпретироваться как вызов функции `wx.NamedColour()`.
`GetId()` `SetId(int)`	Возвращает или устанавливает идентификатор виджета wxPython.
`GetMenuBar()` `SetMenuBar(wx.MenuBar)`	Возвращает или устанавливает используемый фреймом в данный момент объект строки меню, или значение `None`, если строки меню нет.

`GetPosition()`
`GetPositionTuple()`
`SetPosition(wx.Point)`
	Возвращает как объект `wx.Point` или как кортеж Python позицию x, y верхнего левого угла окна. Для окна верхнего уровня эта позиция выражается в дисплейных координатах, а для дочернего окна – в координатах родительского окна.

`GetSize()`
`GetSizeTuple()`
`SetSize(wx.Size)`
	C++ версии методов чтения (getter) и установки (setter) перегружены и используют объект `wx.Size`. Метод `GetSizeTuple()` возвращает размер фрейма в виде кортежа Python. Смотрите также метод `SetDimensions()`, предлагающий другие способы доступа к этой информации.

`GetTitle()` `SetTitle(String)`	Если фрейм был создан со стилем `wx.CAPTION`, связанная с фреймом строка будет отображаться в его полосе заголовка.

Таблица 8.5 приводит некоторые наиболее полезные методы `wx.Frame`. Обратите внимание на метод `Refresh()`, который Вы можете использовать для ручной перерисовки фрейма.

Таблица 8.5 Методы `wx.Frame`

Метод	Описание
`Center(direction=wx.BOTH)`	Центрирует фрейм (Кстати, также определен полный аналог этого метода с неамериканской орфографией `Centre`). Если параметр `direction` будет иметь значение `wx.BOTH`, фрейм центрируется в обоих направлениях, а при значениях `wx.HORIZONTAL` или `wx.VERTICAL`, только в одном направлении.
`Enable(enable=true)`	Если параметр равен `True`, фрейм может принимать ввод пользователя. При значении `False` ввод пользователя во фрейме блокируется. Смежный метод - `Disable()`.
`GetBestSize()`	Возвращает для `wx.Frame` минимальный размер фрейма, необходимый для размещения всех его дочерних окон.
`Iconize(iconize)`	Если параметр `True`, минимизирует фрейм в иконку (точное поведение, конечно же, зависит от системы). Если параметр равен `False`, иконизированный фрейм восстанавливается в нормальное состояние.
`IsEnabled()`	Возвращает `True`, если фрейм в данный момент доступен.
`IsFullScreen()`	Возвращает `True`, если фрейм отображается в полноэкранном режиме, иначе возвращает `False`. Смотри подробности в `ShowFullScreen`.
`IsIconized()`	Возвращается `True`, если фрейм в данный момент иконизирован, и `False` в противном случае.
`IsMaximized()`	Возвращается `True`, если фрейм в данный момент находится в максимизированном состоянии, в противном случае возвращается `False`.
`IsShown()`	Возвращает `True`, если фрейм в данный момент видимый.
`IsTopLevel()`	Всегда возвращает `True` для высокоуровневых виджетов, например для фреймов или диалогов, и `False` для других типов виджетов.

`Maximize(maximize)`	Если параметр равен `True`, фрейм максимизируется и заполняет экран (точное поведение зависит от конкретной системы). При этом происходит тоже самое, что и при нажатии кнопки фрейма «Развернуть», которая обычно расширяет фрейм, так что он заполняет рабочий стол, но не скрывает панель задач и другие системные компоненты.
`Refresh(eraseBackground=True, rect=None)`	Инициирует для фрейма событие перерисовки. Если `rect` имеет значение `none`, фрейм перерисовывается целиком. Если прямоугольник указан, перерисовывается только данная прямоугольная область фрейма. Если `eraseBackground` равен `True`, также будет перерисован и фон окна, но при его значении `False` фон не перерисовывается.
`SetDimensions(x, y, width, height, sizeFlags=wx.SIZE_AUTO)`	Позволяет Вам задать размер и позицию окна при помощи единственного вызова метода. Позиция передается параметрами `x` и `y`, а размер `width` и `height`. Значение -1 любого из первых четырех параметров интерпретируется в зависимости от величины параметра `sizeFlags`. Возможные значения параметра `sizeFlags` содержит таблица 8.6.
`Show(show=True)`	Если передается значение `True`, фрейм будет отображен. При значении `False` фрейм будет скрыт. Метод `Show(False)` эквивалентен методу `Hide()`.
`ShowFullScreen(show, style=wx.FULLSCREEN_ALL)`	Если булевый параметр `show` равен `True`, фрейм отображается в полноэкранном режиме, при котором он расширяется на весь экран, включая перекрытие панели задач и других системных компонентов рабочего стола. Если этот параметр равен `False`, фрейм восстанавливает свой нормальный размер. Параметр `style` является битовой маской. Его значение по умолчанию `wx.FULLSCREEN_ALL` указывает wxPython скрыть все стилевые элементы окна при полноэкранном режиме. Для подавления отдельных частей фрейма в полноэкранном режиме при помощи побитовых операций могут быть составлены другие значения: `wx.FULLSCREEN_NOBORDER`, `wx.FULLSCREEN_NOCAPTION`, `wx.FULLSCREEN_NOMENUBAR`, `wx.FULLSCREEN_NOSTATUSBAR`, `wx.FULLSCREEN_NOTOOLBAR`.

Описанный в таблице 8.5 метод `SetDimensions()` для определения поведения по умолчанию в случае, когда пользователь указывает в качестве размеров величину -1, использует размерные флаги битовой маски. Эти флаги описывает таблица 8.6.

Эти методы не относятся к теме расположения включенных во фрейм конкретных дочерних элементов. Такая тема для более полного описания требует отдельного раздела.

Таблица 8.6 Размерные флаги для метода `SetDimensions`

Флаг	Значение -1 интерпретируется как
`wx.ALLOW_MINUS_ONE`	действительная позиция или размер
`wx.SIZE_AUTO`	преобразовано к значению по умолчанию wxPython
`wx.SIZE_AUTO_HEIGHT`	действительная ширина или высота по умолчанию wxPython
`wx.SIZE_AUTO_WIDTH`	действительная высота или значение ширины по умолчанию wxPython
`wx.SIZE_USE_EXISTING`	текущее значение должно быть перенесено

8.2.2 *Как найти размещенный на фрейме виджет?*

Иногда Вам необходимо найти определенный виджет на фрейме или в панели, уже не имея ссылки на этот виджет. Как показано в главе 6, это обычно применяется, чтобы найти фактический объект пункта меню, связанный с выбором в меню (когда событие уже на него не ссылается). Другое возможное применение, когда Вы хотите, чтобы событие в определенном элементе изменяло состояние другого произвольного виджета в системе. Например, у Вас может быть кнопка и пункт меню, которые взаимно переключают состояние друг друга. Когда кнопка нажимается, Вам требуется получить пункт меню, чтобы выполнить его переключение.

Листинг 8.6 показывает небольшой пример, взятый из главы 7. В этом коде для получения пункта меню, ID которого предоставляет объект-событие, использован метод `FindItemById()`. Текстовая метка этого пункта используется для управления необходимым изменением цвета.

Листинг 8.6 Функция, которая находит элемент с помощью его ID

```
def OnColor(self, event):
    menubar = self.GetMenuBar()
    itemId = event.GetId()
    item = menubar.FindItemById(itemId)
    color = item.GetLabel()
    self.sketch.SetColor(color)
```

В wxPython имеется три действующих одинаковым образом метода для нахождения суб-виджетов. Эти методы применимы к любому виджету, который используется в качестве контейнера, не только к фреймам, но также и к диалогам и панелям. Вы можете найти суб-виджет с помощью его внутреннего ID wxPython, при помощи имени, переданного конструктору в параметре `name`, или при помощи текстовой метки. Текстовая метка определяет заголовок для тех виджетов, которые его имеют, например, для кнопок и фреймов.

Это такие три метода:

- `wx.FindWindowById(id, parent=None)`
- `wx.FindWindowByName(name, parent=None)`
- `wx.FindWindowByLabel(label, parent=None)`

Во всех трех случаях, для того чтобы ограничивать поиск в определенном подмножестве, может быть использован параметр `parent`, (т.е. это эквивалентно вызову для этого параметра метода `Find`). К тому же, метод `FindWindowByName()` выполняет сначала поиск по параметру `name`, и если он не находит соответствия, он вызывает для поиска метод `FindWindowByLabel()`.

8.2.3 Как создать фрейм с полосой прокрутки?

Полосы прокрутки в wxPython не являются элементом фрейма, а управляются классом `wx.ScrolledWindow`. Вы можете использовать `wx.ScrolledWindow` в любом месте, где используется `wx.Panel`, при этом полосы прокрутки будут сдвигать все элементы, размещенные внутри окна с прокруткой. Рисунок 8.5 и 8.6 показывают прокрутку в действии, до и после её выполнения. Верхняя левая кнопка смещает область просмотра (viewport), а нижняя правая кнопка возвращает её обратно.

В этом разделе мы обсудим то, как создать окно с полосой прокрутки и как обрабатывать в Вашей программе выполнение прокрутки.

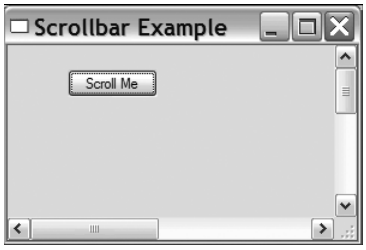

Рисунок 8.5 wx.ScrolledWindow после создания

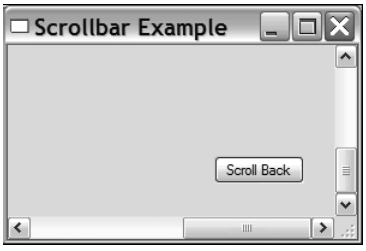

Рисунок 8.6 То же самое окно после прокрутки

Как создать полосу прокрутки

Листинг 8.7 показывает код, использованный для создания окна с прокруткой.

Листинг 8.7 Создание простого окна с прокруткой

```
import wx

class ScrollbarFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Scrollbar Example',
                size=(300, 200))
        self.scroll = wx.ScrolledWindow(self, -1)
        self.scroll.SetScrollbars(1, 1, 600, 400)
        self.button = wx.Button(self.scroll, -1, "Scroll Me",
                pos=(50, 20))
        self.Bind(wx.EVT_BUTTON, self.OnClickTop, self.button)
        self.button2 = wx.Button(self.scroll, -1, "Scroll Back",
                pos=(500, 350))
        self.Bind(wx.EVT_BUTTON, self.OnClickBottom, self.button2)

    def OnClickTop(self, event):
        self.scroll.Scroll(600, 400)

    def OnClickBottom(self, event):
        self.scroll.Scroll(1, 1)

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

Конструктор `wx.ScrolledWindow` почти идентичен конструктору `wx.Panel`:

```
wx.ScrolledWindow(parent, id=-1, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=wx.HSCROLL | wx.VSCROLL,
        name="scrolledWindow")
```

Все его атрибуты ведутся себя, как Вы того ждете, и только атрибут `size` является физическим размером панели в пределах своего родителя, а не логическим размером окна прокрутки.

Определение размера области прокрутки

Имеется несколько автоматизированных методов определения размера области прокрутки. Ручной способ, как показано в листинге 8.1, использует метод `SetScrollBars`:

```
SetScrollbars(pixelsPerUnitX, pixelsPerUnitY, noUnitsX, noUnitsY,
        xPos=0, yPos=0, noRefresh=False)
```

Ключевая концепция – это *шаг прокрутки*, определяемый количеством пространства, на которое сдвигается окно за единичное смещение полосы прокрутки (часто называется смещением бегунка, сопоставляемое со страничным смещением). Первые два параметра, `pixelsPerUnitX` и `PixelsPerUnitY`, позволяют Вам задать размер шага прокрутки в обоих направлениях. Вторые два параметра,

noUnitsX и noUnitsY, позволяют Вам задать размер области прокрутки в шагах прокрутки. Другими словами, размер области прокрутки в пикселях это (pixelsPerUnitX * noUnitsX, pixelsPerUnitY * noUnitsY). Листинг 8.7 предотвращает любую потенциальную неразбериху, устанавливая в качестве шага прокрутки один пиксель. Параметры xPos и yPos позволяют Вам установить начальную позицию полос прокрутки количеством шагов прокрутки (не пикселей), а значение True аргумента noRefresh предотвращает автоматическое обновление окна после любой прокрутки, вызванной методом SetScrollbars().

Примечание переводчика: *В общем случае область просмотра больше размера фрейма и простирается за его границы. Собственно по этой причине и возникает необходимость прокрутки. В приводимом примере из листинга 8.7 область прокрутки больше ровно в два раза размера фрейма, а именно 600x400 против 300x200 пикселей соответственно. Увеличивая мышью исходные размеры окна фрейма, в тот момент, когда размер фрейма станет равен размеру области прокрутки, полосы прокрутки пропадут. Надеюсь, эти пояснения облегчат восприятие оригинальной терминологии прокрутки.*

Имеется три дополнительных метода, которые Вы можете использовать для установки размера области прокрутки с последующей отдельной установкой темпа прокрутки (scroll rate). Эти методы могут показаться Вам проще в использовании, поскольку они позволяют задать размеры в явном виде. Ниже приведено, как Вы можете использовать метод окна прокрутки SetVirtualSize(), устанавливая размер непосредственно в пикселях:

```
self.scroll.SetVirtualSize((600, 400))
```

Используя метод FitInside(), Вы можете установить виджеты в области прокрутки так, чтобы окно прокрутки их ограничивало. Этот метод приводит границы окна прокрутки к необходимому минимуму для точного включения всех суб-окон:

```
self.scroll.FitInside()
```

В общем случае FitInside() применяется, когда внутри окна прокрутки имеется только один виджет (например, текстовый блок), и уже установлен логический размер этого виджета. Если бы мы использовали FitInside() в листинге 8.7, была бы создана меньшая область прокрутки, и так как эта область точно соответствовала бы краю нижней правой кнопки, дополнительный краевой зазор отсутствовал бы.

Наконец, если окно прокрутки содержит внутри координатор (sizer), при помощи метода SetSizer() область прокрутки приводится к размеру управляемых координатором виджетов. Такой механизм чаще всего применяется при сложных видах компоновки. Подробную информацию о координаторах смотрите в главе 11.

Для каждого из этих трех механизмов при помощи метода SetScrollRate() необходимо отдельно устанавливать темп прокрутки (scroll rate):

```
self.scroll.SetScrollRate(1, 1)
```

Параметры этого метода определяют размер шага прокрутки в направлениях x и y соответственно. Размер больший нуля разрешает прокрутку в этом направлении.

События панели прокрутки

Обработчики событий от кнопок в листинге 8.7 с помощью метода `Scroll()` программно изменяют позицию полос прокрутки. Этот метод принимает координаты окна прокрутки x и y в виде шагов прокрутки, а не пикселей.

В главе 7 мы пообещали привести список событий, которые Вы можете получать от полос прокрутки, поскольку они также используются и для управления ползунками (slider). Таблица 8.7 приводит список всех событий прокрутки обрабатываемых непосредственно окном прокрутки. Обычно, у Вас не будет необходимости в использовании многих из этих событий до той поры, пока Вы не начнете создавать собственные виджеты.

Таблица 8.7 События полосы прокрутки

Событие	Описание
EVT_SCROLL	Вызывается при инициировании любого события прокрутки
EVT_SCROLL_BOTTOM	Инициируется, когда пользователь перемещает бегунок полосы прокрутки на максимальный предел её диапазона (нижняя или правая сторона, в зависимости от ориентации).
EVT_SCROLL_ENDSCROLL	В случае MS Windows инициируется в конце любого сеанса прокрутки, вызванного протягиванием бегунка мышью или нажатием клавиши.
EVT_SCROLL_LINEDOWN	Инициируется, когда пользователь перемещает бегунок полосы прокрутки на одну линию вниз.
EVT_SCROLL_LINEUP	Инициируется, когда пользователь перемещает бегунок полосы прокрутки на одну линию вверх.
EVT_SCROLL_PAGEDOWN	Пользователь сдвинул бегунок полосы прокрутки на одну страницу вниз.
EVT_SCROLL_PAGEUP	Бегунок полосы прокрутки сдвинулся на одну страницу вверх.
EVT_SCROLL_THUMBRELEASE	Вызывается в конце любого сеанса прокрутки, выполненного пользователем при помощи фактического перетаскивания бегунка полосы прокрутки с помощью мыши.
EVT_SCROLL_THUMBTRACK	Вызывается многократно до тех пор, пока протягивается бегунок.

EVT_SCROLL_TOP	Инициируется, когда пользователь перемещает бегунок полосы прокрутки в минимальный предел её диапазона (верхняя или левая сторона, в зависимости от ориентации).

Точное определение линии и страницы зависит от установленных Вами шагов прокрутки, одна линия – это один шаг прокрутки, а одна страница является числом полных шагов прокрутки, которые умещаются в видимой части прокручиваемого окна. Для каждого перечисленного в таблице события EVT_SCROLL* есть соответствующее событие EVT_SCROLLWIN*, которое генерируется wx.ScrolledWindow в ответ на события от своих полос прокрутки.

Имеется специфичный для wxPython подкласс окна прокрутки - wx.lib.scrolledpanel.ScrolledPanel, позволяющий Вам автоматически устанавливать прокрутку в панелях, которые для управления компоновкой дочерних виджетов используют координатор. Дополнительное преимущество wx.lib.scrolledpanel.ScrolledPanel состоит в том, что он для перемещения между суб-виджетами позволять пользователю использовать клавишу табуляции. При этом панель автоматически выполняет прокрутку, чтобы стал виден следующий сфокусированный виджет. Для использования класса wx.lib.scrolledpanel.ScrolledPanel объявите его подобно окну с прокруткой, а после добавления всех суб-окон вызовите следующий метод:

```
SetupScrolling(self, scroll_x=True, scroll_y=True, rate_x=20,
        rate_y=20)
```

Параметры rate_x и rate_y являются шагами прокрутки окна, при этом класс автоматически установит виртуальный размер, основанный на размере вложенных виджетов, который вычислил координатор.

Помните, что при определении позиции виджета в окне прокрутки, его позиция всегда является физической позицией виджета относительно фактического начала окна прокрутки в дисплейном фрейме, а не логическая позиция виджета относительно виртуального размера фрейма. Это справедливо даже, если виджет стал невидимым. Например, после нажатия на кнопку Scroll Me на рисунке 8.5, кнопка сообщает свою позицию как (-277, -237). Если это не то, что Вам нужно, то при помощи методов CalcScrolledPosition(x, y) и CalcUnscrolledPosition(x, y) можно переключаться между дисплейными и логическими координатами. В каждом случае, после того, как щелчок на кнопке перемещает прокрутку в правый низ, Вы передаете координаты точки, а окно прокрутки возвращает кортеж (x, y), например, как в следующем случае:

```
CalcUnscrolledPostion(-277, -237) returns (50, 20)
```

8.3 Альтернативные типы фреймов

Фреймы не ограничиваются только обычными прямоугольниками с внутренними виджетами, помимо этого они могут иметь и другие формы. К тому же, Вы можете

создавать фреймы MDI, содержащие внутри себя другие фреймы. Или же Вы можете отказаться от полосы заголовка фрейма и при этом все еще сохранять возможность его перетаскивания пользователем.

8.3.1 *Как создать MDI-фрейм?*

Помните, что такое MDI? Многие люди уже не помнят. MDI был новшеством компании Microsoft 90-х годов, которое допускает в приложении множественные дочерние окна, управляемые единственным родительским окном, и по существу обеспечивает каждое приложение отдельным рабочим столом. В большинстве приложений MDI требует, чтобы все окна приложения минимизировались одновременно и относительно остальной части системы поддерживался одинаковый Z-порядок, а это ограничивает применение. Мы рекомендуем использовать MDI только в тех случаях, когда пользователю целесообразно видеть все окна приложения вместе, например, в играх. Рисунок 8.7 показывает типичную среду MDI.

MDI в wxPython под операционными системами Windows поддерживается при помощи типовых виджетов, а в других операционных системах имитацией дочерних окон. Листинг 8.8 показывает в действии простой пример MDI.

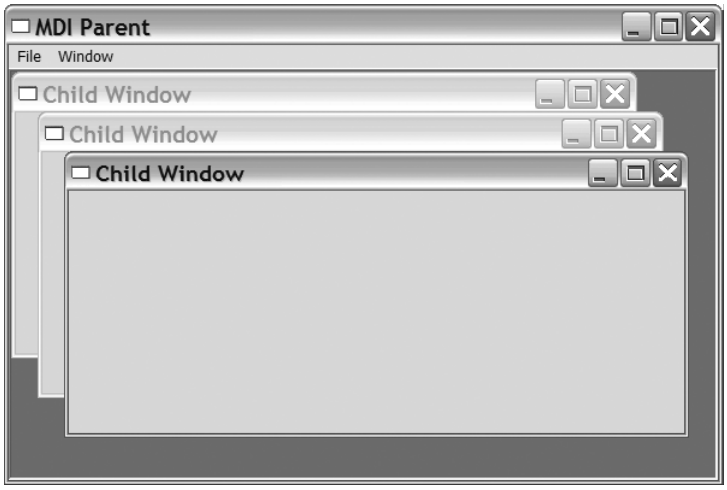

Рисунок 8.7
Окно MDI

Листинг 8.8 Как создать MDI-окно

```
import wx

class MDIFrame(wx.MDIParentFrame):
    def __init__(self):
        wx.MDIParentFrame.__init__(self, None, -1, "MDI Parent",
                size=(600,400))
        menu = wx.Menu()
        menu.Append(5000, "&New Window")
        menu.Append(5001, "E&xit")
        menubar = wx.MenuBar()
```

```
            menubar.Append(menu, "&File")
            self.SetMenuBar(menubar)
            self.Bind(wx.EVT_MENU, self.OnNewWindow, id=5000)
            self.Bind(wx.EVT_MENU, self.OnExit, id=5001)

        def OnExit(self, evt):
            self.Close(True)

        def OnNewWindow(self, evt):
            win = wx.MDIChildFrame(self, -1, "Child Window")
            win.Show(True)

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

Основная концепция MDI довольно проста. Родительское окно является подклассом `wx.MDIParentFrame`, а дочерние окна добавляются точно также как и любой другой виджет wxPython, несмотря на то, что они являются подклассом `wx.MDIChildFrame`. Конструктор `wx.MDIParentFrame` практически идентичен конструктору `wx.Frame`:

```
wx.MDIParentFrame(parent, id, title, pos = wx.DefaultPosition,
        size=wxDefaultSize,
        style=wx.DEFAULT_FRAME_STYLE | wx.VSCROLL | wx.HSCROLL,
        name="frame")
```

Единственное отличие состоит в том, что `wx.MDIParentFrame` имеет по умолчанию прокрутку. Конструктор `wx.MDIChildFrame` аналогичен, но у него нет прокрутки. В листинге 8.8 добавление дочернего фрейма выполнено при помощи его создания, при котором в качестве родителя указан родительский фрейм.

Используя методы родительского фрейма `Cascade()` или `Tile()`, дублирующие одноименные пункты основного меню, Вы можете одновременно изменять позицию и размер всех дочерних фреймов. Вызов `Cascade()` заставляет окна отображаться одно за другим, как на рисунке 8.7, а метод `Tile()` приводит все окна к одинаковому размеру и перемещает их так, чтобы они не перекрывались. Для того чтобы программно переключать фокус между дочерних окон, используйте методы родительского фрейма `ActivateNext()` и `ActivatePrevious()`.

8.3.2 Что такое мини-фрейм и когда он применяется?

Мини-фрейм (mini-frame) идентичен обычному фрейму, за исключением двух важных особенностей: он имеет меньшую область заголовка и не отображается в панели задачи MS Windows или GTK. Рисунок 8.8 показывает пример меньшей области заголовка.

Рисунок 8.8 Мини-фрейм в действии

Код для создания мини-фрейма почти такой же, как и код создания обычного фрейма, единственное отличие состоит в том, что в качестве родительского класса теперь выступает `wx.MiniFrame`. Данный код приведен в листинге 8.9.

Листинг 8.9 Создание мини-фрейма

```
import wx

class MiniFrame(wx.MiniFrame):
    def __init__(self):
        wx.MiniFrame.__init__(self, None, -1, 'Mini Frame',
                size=(300, 100))
        panel = wx.Panel(self, -1, size=(300, 100))
        button = wx.Button(panel, -1, "Close Me", pos=(15, 15))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)

    def OnCloseMe(self, event):
        self.Close(True)
    def OnCloseWindow(self, event):
        self.Destroy()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    MiniFrame().Show()
    app.MainLoop()
```

Конструктор для `wx.MiniFrame` идентичен конструктору для `wx.Frame`, однако мини-фрейм поддерживает дополнительные стилевые флаги, которые перечислены в таблице 8.8.

Таблица 8.8 Стилевые флаги `wx.MiniFrame`

Событие	Описание
`wx.THICK_FRAME`	В MS Windows и Motif рисует фрейм с толстой границей.
`wx.TINY_CAPTION_HORIZONTAL`	Замещает `wx.CAPTION`, отображая уменьшенный горизонтальный заголовок.
`wx.TINY_CAPTION_VERTICAL`	Замещает `wx.CAPTION`, отображая уменьшенный вертикальный заголовок.

Обычно, мини-фреймы используются для инструментальных окон (в стиле Photoshop), где они всегда доступны и не загромождают панель задач. Уменьшенный заголовок позволяет им эффективнее использовать пространство и визуально отделяет их от обычных фреймов.

8.3.3 Как создать непрямоугольный фрейм?

Поскольку прямоугольники имеют точную правильную форму и относительно просты для прорисовки и поддержки, в большинстве приложений используются именно прямоугольные фреймы. Иногда, все же, Вам необходимо разрывать оковы прямых линий. В wxPython Вы можете придать фрейму произвольную форму. Если определена альтернативная форма, то части фрейма за пределами этой формы не прорисовываются, не реагируют на события мыши и даже не принадлежат данному фрейму. Рисунок 8.9 показывает пример формованного окна, отображённого на фоне кода текстового редактора.

События настроены таким образом, что двойной щелчок включает и выключает нестандартную форму, а щелчок правой кнопкой закрывает окно. Этот пример использует в качестве источника для картинки Vippi (это талисман wxPython) модуль `images` демонстрационного набора wxPython.

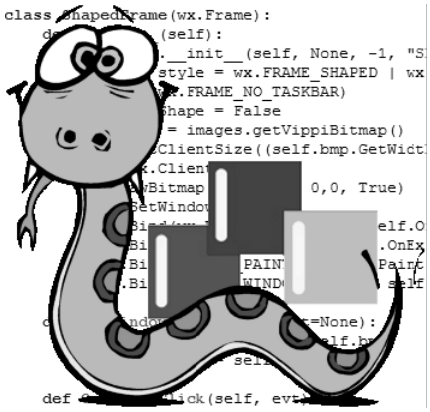

Рисунок 8.9
Окно непрямоугольной формы

Листинг 8.10 отображает код, который находится за этим непрямоугольным фреймом (наверняка, Вы затрудняетесь его прочитать за талисманом на рисунке 8.9). Этот пример чуть сложнее, чем другие рассмотренные ранее, так как показывает, как управлять вещами, подобными закрытию окна в отсутствие его штатных элементов интерфейса.

Листинг 8.10 Рисование окна произвольной формы

```
import wx
import images

class ShapedFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Shaped Window",
                style = wx.FRAME_SHAPED | wx.SIMPLE_BORDER |
                wx.FRAME_NO_TASKBAR)
        self.hasShape = False
        self.bmp = images.getVippiBitmap()
        self.SetClientSize((self.bmp.GetWidth(),
            self.bmp.GetHeight()))
```

1 **Получение изображения**

```
        dc = wx.ClientDC(self)
        dc.DrawBitmap(self.bmp, 0,0, True)
        self.SetWindowShape()
        self.Bind(wx.EVT_LEFT_DCLICK, self.OnDoubleClick)
        self.Bind(wx.EVT_RIGHT_UP, self.OnExit)
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_WINDOW_CREATE, self.SetWindowShape)

    def SetWindowShape(self, evt=None):
        r = wx.RegionFromBitmap(self.bmp)
        self.hasShape = self.SetShape(r)

    def OnDoubleClick(self, evt):
        if self.hasShape:
            self.SetShape(wx.Region())
            self.hasShape = False
        else:
            self.SetWindowShape()

    def OnPaint(self, evt):
        dc = wx.PaintDC(self)
        dc.DrawBitmap(self.bmp, 0,0, True)

    def OnExit(self, evt):
        self.Close()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    ShapedFrame().Show()
    app.MainLoop()
```

2 Рисование изображения

3 Подключение события создания окна

4 Установка формы

5 Переключение формы

1. После получения изображения из модуля `images` мы привели размер внутренней части окна к размеру этого битового изображения. Вы также можете создать битовое изображение wxPython из обычного графического файла, что будет обсуждаться подробнее в главе 16.
2. В данном случае, мы рисуем изображение в окне. Это является обязательным требованием. Точно также как и в любом другом окне, Вы можете разместить в формованном окне виджеты и текст (только они должны находиться в пределах пространства формы этого окна).
3. Это событие приводит к вызову `SetWindowShape()` после создания окна, хотя на большинстве платформ в нем нет необходимости. Тем не менее, реализация GTK требует, чтобы родной UI-объект окна был создан и готов прежде, чем будет установлена форма, поэтому мы используем событие создания окна, определяющее, когда это происходит, и устанавливаем форму в его обработчике.
4. Мы используем глобальный метод `wx.RegionFromBitmap`, чтобы создать объект `wx.Region`, необходимый для установки формы. Это простейший способ создания нестандартной формы. К тому же, Вы можете создать `wx.Region` с помощью определяющего многоугольник списка точек. В качестве границ, определяющих область формы, используется прозрачная часть маски

изображения.

5. Событие двойного щелчка переключает форму окна. Для того чтобы вернуть форму в нормальный прямоугольный вид, вызовите метод `SetShape()`, используя в качестве аргумента пустой объект `wx.Region`.

8.3.4 Как перетягивать фрейм, не имеющий заголовка?

Очевидный результат предыдущего примера состоит в том, что фрейм обездвижен – у него отсутствует полоса заголовка и стандартный метод его перетаскивания в этом случае недоступен. Чтобы решить эту проблему, нам нужно добавить обработчик события, который будет двигать окно при операции перетаскивания. Листинг 8.11 выводит такое же формованное окно, как и в прежнем примере, но с некоторыми дополнительными событиями для обработки щелчков левой кнопки мыши и её перемещений. Эта техника применима к любому другому фрейму или даже к перемещаемому внутри фрейма окну (например, некоторому элементу чертёжной программы).

Листинг 8.11 События, позволяющие перетаскивать фрейм за его тело

```
import wx
import images

class ShapedFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Shaped Window",
                style = wx.FRAME_SHAPED | wx.SIMPLE_BORDER )
        self.hasShape = False
        self.delta = wx.Point(0,0)
        self.bmp = images.getVippiBitmap()
        self.SetClientSize((self.bmp.GetWidth(),
            self.bmp.GetHeight()))
        dc = wx.ClientDC(self)
        dc.DrawBitmap(self.bmp, 0,0, True)
        self.SetWindowShape()
        self.Bind(wx.EVT_LEFT_DCLICK, self.OnDoubleClick)
        self.Bind(wx.EVT_LEFT_DOWN, self.OnLeftDown)
        self.Bind(wx.EVT_LEFT_UP, self.OnLeftUp)
        self.Bind(wx.EVT_MOTION, self.OnMouseMove)
        self.Bind(wx.EVT_RIGHT_UP, self.OnExit)
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_WINDOW_CREATE, self.SetWindowShape)

    def SetWindowShape(self, evt=None):
        r = wx.RegionFromBitmap(self.bmp)
        self.hasShape = self.SetShape(r)

    def OnDoubleClick(self, evt):
        if self.hasShape:
            self.SetShape(wx.Region())
            self.hasShape = False
```

1 Новые события

```
            else:
                self.SetWindowShape()

        def OnPaint(self, evt):
            dc = wx.PaintDC(self)
            dc.DrawBitmap(self.bmp, 0,0, True)

        def OnExit(self, evt):
            self.Close()

        def OnLeftDown(self, evt):
            self.CaptureMouse()
            pos = self.ClientToScreen(evt.GetPosition())
            origin = self.GetPosition()
            self.delta = wx.Point(pos.x - origin.x, pos.y - origin.y)

        def OnMouseMove(self, evt):
            if evt.Dragging() and evt.LeftIsDown():
                pos = self.ClientToScreen(evt.GetPosition())
                newPos = (pos.x - self.delta.x, pos.y - self.delta.y)
                self.Move(newPos)

        def OnLeftUp(self, evt):
            if self.HasCapture():
                self.ReleaseMouse()

    if __name__ == '__main__':
        app = wx.PySimpleApp()
        ShapedFrame().Show()
        app.MainLoop()
```

1. Для выполнения всей работы мы добавляем обработчики трех событий мыши: нажатие левой кнопки, отпускание левой кнопки и перемещение мыши.
2. Событие перетаскивания стартует при нажатии левой кнопки мыши. Этот обработчик события выполняет две вещи. Во-первых, он захватывает мышь, что предотвращает посылку событий мыши другим виджетам до тех пор, пока мышь не будет отпущена. Во-вторых, он вычисляет смещение между позицией, где произошло событие и верхним левым углом окна. Это смещение будет использовано при перемещении мыши для вычисления новой позиции окна.
3. Этот обработчик вызывается при перемещении мыши, он сначала проверяет, имеется ли событие перетаскивания, совмещенное с нажатой левой кнопкой. Если это так, он использует новую позицию мыши и ранее рассчитанное смещение, чтобы определять новую позицию окна и выполнить его перемещение.
4. Когда левая кнопка мыши отпускается, вызов `ReleaseMouse()` снова разрешает отсылку событий мыши другим виджетам.

Чтобы удовлетворять любым иным требованиям, данная техника перетаскивания может быть соответствующим образом адаптирована. Например, если щелчок

мыши должен начинать перетаскивание, когда он производится в пределах определенной области, Вам необходимо в событии нажатия мыши выполнить проверку начальной позиции и разрешать перетаскивание, если щелчок произведен в правильном месте.

8.4 Использование оконных разделителей (splitter)

Окно-разделитель (splitter window) – это особый тип контейнерного виджета, который управляет ровно двумя подчинёнными окнами (суб-окна). Эти два суб-окна могут составляться горизонтально или рядом друг с другом с левой или правой стороны. Между двумя суб-окнами размещается разделительная полоса (sash), которая является подвижной границей, изменяющей размер этих двух суб-окон. Окна-разделители часто используются в основном окне для разворачиваемых боковых панелей (например, браузеров объектов). Рисунок 8.10 показывает пример окна-разделителя.

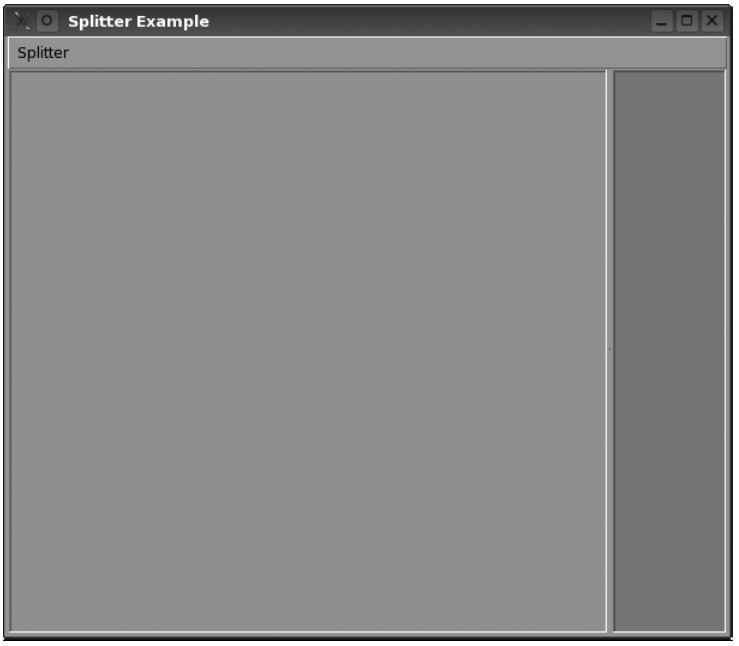

Рисунок 8.10 Пример окна-разделителя после инициализации

Окна-разделители полезны, когда у Вас имеется две информационные панели, и Вы хотите, чтобы пользователь независимо устанавливал размер каждой из них. Примером оконного разделителя может служить окно Mac OS X Finder, а многие текстовые редакторы или графические программы для поддержания списка открытых файлов используют нечто подобное.

8.4.1 Создание оконного разделителя

В wxPython окно-разделитель является экземпляром класса `wx.SplitterWindow`. В отличие от большинства других виджетов wxPython, окна-разделители после их создания и до момента их использования, требуют дополнительной инициализации. Конструктор окна-разделителя лаконичен и прост:

```
wx.SplitterWindow(parent, id=-1, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=wx.SP_3D,
        name="splitterWindow")
```

Его параметры имеют стандартное значение: `parent` – это контейнер для данного виджета, `pos` – определяет размещение виджета в родителе, `size` – определяет размер окна-разделителя.

После создания окна-разделителя, прежде, чем оно может быть использовано, Вы должны вызвать один из трех его методов. Если Вы хотите, чтобы Ваше окно-разделитель изначально отображалось только с одним суб-окном, вызовите метод `Initialize(window)`, где параметр `window` является отдельным суб-окном (обычно типа `wx.Panel`). В этом случае, окно будет разделено позже, в ответ на некоторое пользовательское действие.

Для выполнения разделения используйте метод `SplitHorizontally (window1, window2, sashPosition=0)` или `SplitVertically(window1, window2, sashPosition=0)`. Оба этих метода работают аналогично. Параметры `window1` и `window2` содержат два суб-окна, а параметр `sashPosition` содержит начальную позицию разделительной полосы. Для горизонтальной версии, `window1` размещается сверху `window2`. Если `sashPosition` - положительное число, оно представляет начальную высоту верхнего окна (или для полосы разделителя - количество пикселей сверху). Если `sashPosition` - отрицательное число, оно определяет размер нижнего окна, или количество пикселей снизу от разделителя. Если `sashPosition` - 0, тогда полоса разделителя проходит точно посредине. В вертикальном методе разделения, `window1` находится слева, а `window2` справа. Кроме того, положительное значение `sashPosition` устанавливает размер `window1` и количество пикселей от левой границы до полосы разделителя. Отрицательный `sashPosition` аналогично устанавливает размер окна справа, а 0 устанавливает разделительную полосу в центре. Если Ваше суб-окно достаточно сложное, мы рекомендуем Вам использовать в его компоновке координаторы, которые позволят при перемещении разделительной полосы изящно реагировать на изменение оконных размеров.

8.4.2 Пример разделителя

Код примера из листинга 8.12 показывает, как разделитель создается в одном суб-окне и выполняет фактическое разделение позже в ответ на выбор меню. Также этот листинг использует некоторые события, о которых мы поговорим позднее. Заметьте то, как скрывается при вызове метода `Hide()` суб-панель, которую мы вначале не планируем делать в разделителе видимой. Мы скрываем эту суб-

панель, поскольку в исходном состоянии не требуем от разделителя управлять её размером и размещением. Если бы мы изначально выполнили разделение и показывали обе суб-панели, об этом не следовало бы беспокоиться.

Листинг 8.12 Как создать собственное окно-разделитель

```
import wx

class SplitterExampleFrame(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, title=title)
        self.MakeMenuBar()
        self.initpos = 100
        self.sp = wx.SplitterWindow(self)
        self.p1 = wx.Panel(self.sp, style=wx.SUNKEN_BORDER)
        self.p2 = wx.Panel(self.sp, style=wx.SUNKEN_BORDER)
        self.p2.Hide()
        self.p1.SetBackgroundColour("pink")
        self.p2.SetBackgroundColour("sky blue")
        self.sp.Initialize(self.p1)
        self.sp.SetMinimumPaneSize(10)

    def MakeMenuBar(self):
        menu = wx.Menu()
        item = menu.Append(-1, "Split horizontally")
        self.Bind(wx.EVT_MENU, self.OnSplitH, item)
        self.Bind(wx.EVT_UPDATE_UI, self.OnCheckCanSplit, item)
        item = menu.Append(-1, "Split vertically")
        self.Bind(wx.EVT_MENU, self.OnSplitV, item)
        self.Bind(wx.EVT_UPDATE_UI, self.OnCheckCanSplit, item)
        item = menu.Append(-1, "Unsplit")
        self.Bind(wx.EVT_MENU, self.OnUnsplit, item)
        self.Bind(wx.EVT_UPDATE_UI, self.OnCheckCanUnsplit, item)
        menu.AppendSeparator()
        item = menu.Append(wx.ID_EXIT, "E&xit")
        self.Bind(wx.EVT_MENU, self.OnExit, item)
        mbar = wx.MenuBar()
        mbar.Append(menu, "Splitter")
        self.SetMenuBar(mbar)

    def OnSplitH(self, evt):
        self.sp.SplitHorizontally(self.p1, self.p2, self.initpos)

    def OnSplitV(self, evt):
        self.sp.SplitVertically(self.p1, self.p2, self.initpos)

    def OnCheckCanSplit(self, evt):
        evt.Enable(not self.sp.IsSplit())

    def OnCheckCanUnsplit(self, evt):
        evt.Enable(self.sp.IsSplit())

    def OnUnsplit(self, evt):
        self.sp.Unsplit()
```

Создание окна-разделителя

Создание суб-панелей

Скрывает запасную суб-панель

Инициализация разделителя

Отклик на запрос о горизонтальном разделении

Отклик на запрос о вертикальном разделении

```
    def OnExit(self, evt):
        self.Close()

app = wx.PySimpleApp(redirect=True)
frm = SplitterExampleFrame(None, "Splitter Example")
frm.SetSize((600,500))
frm.Show()
app.SetTopWindow(frm)
app.MainLoop()
```

Окно-разделитель может быть разделено только однократно. Попытка разделения уже разделенного окна потерпит неудачу, и как результат в методе разделения, возвращается `False` (при успехе возвращается `True`). Для определения того, разделено ли окно-разделитель в данный момент, вызовите метод `IsSplit()`. В листинге 8.12 это сделано для подтверждения доступности соответствующих пунктов меню.

Если Вы хотите отменить разделение окна, используйте метод `Unsplit(toRemove=None)`. Параметр `toRemove` представляет реальный объект `wx.Window` для удаления, и должен являться одним из двух суб-окон. Если `toRemove` равен `None`, то в зависимости от ориентации разделителя, удаляется нижнее или правое окно. По умолчанию, удаленное окно wxPython не уничтожает, поэтому Вы можете позднее вернуть его назад. При успешном снятии разделения метод `Unsplit` возвращает `True`. Если окно-разделитель в данный момент не разделено, или если аргумент `toRemove` не является одним из суб-окон данного разделителя, возвращается `False`.

Для подтверждения, что Вы имеете реальную ссылку на под-окно, можете использовать методы получения `GetWindow1()` и `GetWindow2()`. Метод `GetWindow1()` возвращает верхнее или левое суб-окно, а `GetWindow2()` возвращает нижнее или правое суб-окно. Поскольку прямого метода установки суб-окна не существует, используйте для его замещения метод `ReplaceWindow(winOld, winNew)`, где `winOld` - замещаемый Вами объект `wx.Window`, а `winNew` - новое отображаемое окно.

8.4.3 *Изменение внешнего вида разделителя*

Внешним видом окна-разделителя управляет множество стилевых флагов. Заметьте, так как разделитель пытается нарисовать свою полосу в стиле, соответствующем элементам управления исходной платформы, не все из перечисленных стилевых флагов будут работать в разных системах. Таблица 8.9 описывает доступные флаги.

Как мы увидим в следующем разделе, Вы можете также изменить вид разделителя из Вашего приложения – в ответ на действия пользователя, или же основываясь на Ваших собственных предпочтениях.

Таблица 8.9 Стилевые флаги окна-разделителя

Стиль	Описание
`wx.SP_3D`	Рисует границу окна и полосу разделителя с эффектом 3D. Это стиль по умолчанию.
`wx.SP_3DBORDER`	Рисует с эффектом 3D только границу окна, но не полосу разделителя.
`wx.SP_3DSASH`	Рисует с эффектом 3D только полосу разделителя, но не границу.
`wx.SP_BORDER`	Рисует границу вокруг окна без 3D.
`wx.SP_LIVE_UPDATE`	Изменяет встроенное поведение при отклике на перемещение разделительной полосы. Если этот флаг не установлен, то пока пользователь выполняет перетаскивание разделителя, рисуется линия, которая указывает новую позицию разделительной полосы. Размеры суб-окон обновятся только после окончания перетаскивание разделительной полосы. Если же этот флаг установлен, тогда при перетаскивании разделительной полосы оба суб-окна непрерывно изменяют свои размеры, положение и перерисовываются.
`wx.SP_NOBORDER`	Вообще не рисует границы.
`wx.SP_NO_XP_THEME`	В Windows XP не использует для полосы разделителя тему XP, придавая окну классический вид.
`wx.SP_PERMIT_UNSPLIT`	Если этот флаг установлен, с окна может быть всегда снято разделение (Unsplit). Если не установлен, Вы можете предохранить окно от снятия разделения путем назначения минимальному размеру панели значения больше нуля.

8.4.4 Программное управление разделителем

Как только окно-разделитель будет создано, для манипуляции позицией разделительной полосы Вы можете использовать методы этого окна. А именно, для перемещения разделительной полосы Вы можете использовать метод `SetSashPosition(position, redraw=True)`. Параметр `position` является её новой позицией в пикселях, отсчитываемой для горизонтальной разделительной полосы сверху, или для вертикальной полосы слева. Отрицательные значения используются так же, как и в методах разделения для указания позиции с другой стороны. Если параметр `redraw` равен `True`, окно обновляется немедленно, в противном случае оно ожидает обычного оконного обновления. Если указанное значение пикселей лежит вне диапазона, поведение данного установочного метода не определено. Для получения текущей позиции разделительной полосы используйте метод `GetSashPosition()`.

Поведение разделителя по умолчанию позволяет пользователю перемещать разделительную полосу на всём пространстве между двумя границами окна. Перемещение разделительной полосы к одной границе сводит к нулю размер одного из суб-окон, что де-факто снимает с окна разделение. Чтобы этого избежать, Вы можете определить минимальный размер суб-окна при помощи метода `SetMinimumPaneSize(paneSize)`. Параметр `paneSize` является минимальным размером суб-окна в пикселях. Для создания миниатюрных суб-окон пользователь лишен возможности перетаскивания разделительной полосы слишком далеко, поэтому аналогичным образом ограничено и программное изменение позиции разделительной полосы. Как упомянуто в этой главе ранее, объявляя окно со стилем `wx.SP_PERMIT_UNSPLIT`, Вы можете разрешить программное снятие разделения (unsplit) даже при минимальном размере суб-окна. Для того чтобы получать текущий минимальный размер суб-окна, используйте метод `GetMinimumPaneSize()`.

Способ разделения окна изменяется методом `SetSplitMode(mode)`, где параметр `mode` – это одна из констант `wx.SPLIT_VERTICAL` или `wx.SPLIT_HORIZONTAL`. При изменении способа разделения верхнее окно становится левым, а нижнее правым (и наоборот, при другом способе разделения). Этот метод не вызывает перерисовку окна, вместо этого, Вы должны её выполнить явно. Определить текущий способ разделения Вы можете с помощью метода `GetSplitMode()`, который возвращает одно из двух приведенных выше константных значений. Если окно в данный момент не разделено, метод `GetSplitMode()` возвращает самый последний его способ разделения.

Обычно, если не установлен стиль `wx.SP_LIVE_UPDATE`, суб-окно изменяет размеры только в конце сеанса перетаскивания разделительной полосы. Если же Вы хотите заставить суб-окно выполнить перерисовку в любое другое время, используйте метод `UpdateSize()`.

8.4.5 *Обработка событий разделителя*

Окна-разделители инициируют события класса `wx.SplitterEvent`. Как указано в таблице 8.10, окно-разделитель имеет четыре различных типов событий.

Таблица 8.10 Типы событий окна-разделителя

Тип события	Описание
`EVT_SPLITTER_DCLICK`	Инициируется при двойном щелчке на разделительной полосе. До тех пор, пока вы не вызовете метод события `Veto()`, обработка этого события не блокирует выполняемого в ответ нормального снятия разделения.

EVT_SPLITTER_SASH_POS_CHANGED	Инициируется в конце перемещения разделительной полосы, но прежде, чем изменение будет отображено на экране (поэтому Вы можете реагировать на это). Это событие может быть также приостановлено при помощи метода `Veto()`.
EVT_SPLITTER_SASH_POS_CHANGING	Инициируется многократно при перетаскивании разделительной полосы. Это событие может быть приостановлено при помощи метода события `Veto()`, в этом случае позиция разделительной полосы не изменяется.
EVT_SPLITTER_UNSPLIT	Данное событие инициируется после того, как разделение было снято.

Класс инициируемых разделителем событий является подклассом `wx.CommandEvent`. Экземпляр этого класса предоставляет Вам доступ к информации о текущем состоянии окна-разделителя. Для двух событий, касающихся перемещения разделительной полосы, вызов метода `GetSashPosition()` возвращает позицию разделительной полосы относительно левой или верхней границы окна, в зависимости от ориентации разделителя. В событии, многократно происходящем при изменении позиции (`EVT_SPLITTER_SASH_POS_CHANGING`), вызов метода `SetSashPosition(pos)` и `XOR` отслеживает путь, показывая, что ожидаемое положение разделительной полосы перемещается в новую позицию. В событии, происходящем после изменения позиции (`EVT_SPLITTER_SASH_POS_CHANGED`), тот же метод переместит полосу разделителя непосредственно. Для события двойного щелчка Вы можете получить точную позицию щелчка, используя методы события `GetX()` и `GetY()`. С помощью метода `GetWindowBeingRemoved()` в событии снятия разделения (`EVT_SPLITTER_UNSPLIT`) Вы можете узнать, какое окно уходит.

8.5 Резюме

- Значительная часть действий пользователя в программе wxPython происходит внутри объектов `wx.Frame` или `wx.Dialog`. Класс `wx.Frame` представляет все то, что пользователь обычно называет окном. Экземпляры `wx.Frame` создаются почти таким же способом, как и другие виджеты wxPython. Типичное использование `wx.Frame` включает создание подкласса, который расширяет базовый класс, как правило, определяя суб-виджеты, компоновку и поведение. Обычно фрейм содержит единственный высокоуровневый суб-виджет типа `wx.Panel` или какое-то другое контейнерное окно.
- `wx.Frame` имеет ряд специфических стилевых флагов. Некоторые из этих флагов влияют на размер и форму фрейма, другие влияют на то, как он отображается относительно других фреймов системы, другие же флаги определяют, какие элементы интерфейсного оформления находятся на

фреймовой границе. В некоторых случаях для определения стилевого флага необходим двухступенчатый процесс.

- При помощи метода `Close()` может быть выполнен запрос на закрытие фрейма. Это дает фрейму возможность закрыть любые удерживаемые им ресурсы. Фрейм может также наложить вето на запрос о закрытии. Вызов метода `Destroy()` заставляет фрейм незамедлительно закрыться.
- Определенный суб-виджет на фрейме может быть найден с помощью ID wxPython, имени или текстовой метки.
- Прокрутка обеспечивается включением контейнерного виджета типа `wx.ScrolledWindow`. Имеется несколько способов установить параметры прокрутки, простейший способ использует в окне с прокруткой координатор. В этом случае wxPython автоматически определяет виртуальный размер панели прокрутки. Тем не менее, если требуется, виртуальный размер можно установить вручную.
- Имеется пара различных фреймовых подклассов, которые учитывают другие взгляды. Класс `wx.MDIParentFrame` может быть использован для создания MDI, а `wx.MiniFrame` может создать окна в инструментальном стиле с уменьшенной полосой заголовка. При помощи метода `SetShape()` можно придать фреймам непрямоугольную форму. Внутренняя область фрейма (регион) может быть задана с помощью любого битового изображения, с простой цветовой маской, позволяющей определить края региона. Непрямоугольное окно обычно не имеет полосы заголовка, допускающей перетаскивание фрейма, но этим можно управлять, явно обрабатывая события мыши.
- При помощи класса `wx.SplitterWindow` между двумя суб-окнами может быть реализована перетаскиваемая разделительная полоса, которая может управляться пользователем в диалоговом режиме или при необходимости программно.

В следующей главе мы обсудим диалоговые панели, которые ведут себя подобно фреймам.

Вы можете поддержать перевод этой и других книг, выполнив небольшое дружественное денежное перечисление на счет автора перевода в одной из указанных ниже систем электронных платежей. Размер перечисления определяйте, исходя из ценности получаемой Вами информации. Каждый Ваш взнос с благодарностью принимается в качестве поощрения проделанного труда.

http://ukrmoney.com
Идентификатор пользователя: YSavitskiy@ukr.net (идентификатором счета выступает e-mail)

http://webmoney.ru
U701373884420, E642860582665, R222115441462, Z354483832040

В любом случае, все переведенные главы данной книги всегда доступны для свободной загрузки с сайта python.com.ua в разделе «Переводы» (http://python.com.ua/translate/wxpython/).