Перевод: Володеев Сергей

Это не профессиональный перевод. Возможны ошибки и неточности. Если какой-то фрагмент вызывает у вас сомнения, смотрите английский вариант книги.

Глава 3. Работа в среде, управляемой событиями

Эта глава включает

- Программирование в среде, управляемой событиями
- Связь событий с обработчиками
- Распространение событий в wxPython
- Создание своих событий

Обработка событий - фундаментальный механизм, на котором основана работа программ wxPython. Такие программы называют программами, управляемыми событиями. В этой главе, мы обсудим, чем управляемое событиями приложение отличается от традиционного. Мы дадим краткий обзор понятий и терминологии, используемой в программирование GUI. Мы также расскажем о жизненном цикле типичной управляемой событиями программы.

Событие – это то, что случается в вашей системе, и на что ваше приложение может отреагировать, вызывая определенную функцию. Событие может быть низкоуровневым действием пользователя, типа перемещения мыши или нажатия клавиши, или высокоуровневым, типа выбора из меню. Событие может также быть создано операционной системой. Созданные вами объекты также могут создавать собственные события. Приложение wxPython работает, связывая определенный вид события с определенным программным кодом, который должен быть выполнен в ответ на событие. Такой программный код, связанный с событием, называется обработчиком события.

Эта глава рассказывает о событиях, о том как писать код для обработки событий, и о том как система wxPython вызывает ваш код, когда событие произошло. Мы также покажем вам, как добавить собственные события к библиотеке wxPython, которая содержит список стандартных пользовательских и системных действий.

3.1 Терминология для понимания событий

Эта глава содержит много терминов, большая часть которых начинается со слова событие (event). Таблица 3.1 - справочник терминов, которые мы будем использовать.

Таблица 3.1 Термины, связанные с событием

Термин	**Описание**
event	Событие. Что-то, что случается во время работы вашего приложения, и что требует какой-то реакции.
event object	Событие как объект. События представлены как объекты класса wx.Event и его подклассов, типа wx.CommandEvent и wx.MouseEvent.
event type	Тип события. В wxPython каждое событие имеет целочисленный идентификатор – тип события, который уточняет природу события. Например, у события wx.MouseEvent есть тип, который указывает, что событие является или щелчком мыши или перемещением мыши.
event source	Источник события. Любой объект wxPython, который создал событие. Например: кнопка, пункт меню, список или любой другой виджет.
event-driven	Программа имеющая такую структуру, где большая часть времени отводится

	на ожидание и обработку событий.
event queue	Очередь событий. Список событий, которые уже произошли, но еще не были обработаны.
event handler	Обработчик события. Функция или метод, который вызывается в ответ на событие.
event binder	Биндер. Объект wxPython, который инкапсулирует связь между определенным виджетом, определенным типом события и соответствующим обработчиком. Чтобы быть вызванными, все обработчики событий должны быть зарегистрированы биндерами.
wx.EvtHandler	Класс wxPython, который позволяет его объектам создавать связь между биндером определенного типа, источником события (event source) и обработчиком события (event handler). Отметьте, что класс wx.EvtHandler не то же самое, что функция или метод обработчик события.

Мы надеемся, что эта таблица не позволит вам перепутать обработчики событий с биндерами. Обращайтесь к этой таблице по мере необходимости. Мы начнем с краткого обзора управляемого событиями программирования, затем мы обсудим специфические особенности того, как все это реализовано в wxPython.

3.2 Программирование управляемое событиями

Программа, управляемая событиями, имеет структуру управления, которая, главным образом, получает события и отвечает на них. Структура программы wxPython (или любой другой программы, управляемой событиями) существенно отличается от структуры обычного сценария Python. Типичный сценарий Python имеет определенную отправную точку и определенный пункт окончания, и программист управляет порядком выполнения, используя условные операторы, циклы и функции. Программа не линейна, но ее порядок часто независим от пользовательских действий.

С точки зрения пользователя, программа wxPython большую часть времени ничего не делает. Она ожидает пока пользователь или система проявят свою активность. Структура программы wxPython – это пример архитектуры программы, управляемой событиями. На рисунке 3.1 показаны главные части программы, управляемой событиями.

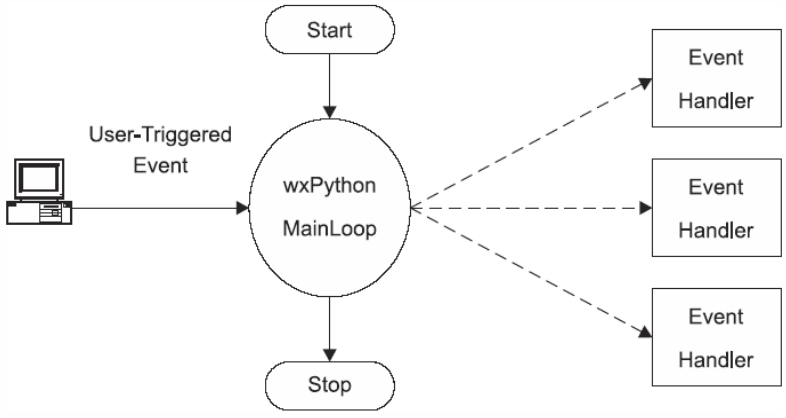

Рисунок 3.1 Схематичное изображение цикла обработки событий. Показан жизненный путь программы, пользовательские события и вызов обработчиков.

Цикл обработки событий можно сравнить с оператором в сервис-центре. Пока нет входящих звонков оператор ожидает. Наконец происходит событие - телефонный звонок. В процессе общения с клиентом, оператор должен получить достаточно информации, чтобы знать к кому переадресовать клиента для ответа. После пересылки, оператор ждет следующего звонка.

Хотя каждая система управляемая событиями имеет свои особенности, между ними много общего. Первичные характеристики программы управляемой событиями следующие:

- После начальной установки, большую часть времени программа проводит в цикле, ожидая события. Вход в этот цикл показывает начало пользовательской интерактивной части программы, и выход из цикла показывает ее конец. В wxPython, этот цикл – метод wx.App.MainLoop(). Он должен быть явно вызван в вашем сценарии. Выход из цикла происходит, когда все окна верхнего уровня закрыты.

- Программа реагирует на события, которые могут произойти в среде программы. Обычно, события вызываются пользовательской деятельностью, но могут также быть результатом деятельности системы, или произвольного кода в другом месте программы. В wxPython, все события – объекты класса wx.Event или производного от него. Каждое событие имеет атрибут тип события (event type) (см. таблицу 3.1), который позволяет различать виды событий.

- В цикле, программа периодически проверяет, случилось ли что-нибудь требующее ответа. Есть несколько механизмов, которыми система, управляемая событиями, может получать информацию о событиях. Наиболее популярный метод, используемый wxPython: события помещаются в общую очередь по мере поступления, а затем извлекаются из нее и обрабатываются.

- Когда событие происходит, система его обрабатывает, вызывая программный код, связанный с этим событием. В wxPython, родные системные события переведены в объекты wx.Event и созданы методы wx.EvtHandler.ProcessEvent() для того, чтобы послать события надлежащему обработчику. На рисунке 3.3 схематически изображен этот процесс. Составляющие части механизма обработки событий – биндеры (event binder) и обработчики событий. Биндер (event binder) - предопределенный объект wxPython. Есть отдельный биндер для каждого типа события. Обработчик (event handler) - функция или метод, который принимает объект события как параметр. Обработчик вызывается, когда пользователь вызывает соответствующее событие.

Далее, мы подробно обсудим как это реализовано в wxPython. И начнем с обработчиков событий.

3.2.1 Программирование обработчиков событий

В вашем коде wxPython, события и обработчики событий должны быть связаны с соответствующими виджетами. Например, событие нажатия кнопки будет послано определенному обработчику, связанному с этой кнопкой. Для связи события, виджета и обработчика используете специальный объект - биндер. Например,

```
self.Bind(wx.EVT_BUTTON, self.OnClick, aButton)
```

используется предопределенный объект биндер wx.EVT_BUTTON для связи события нажатия кнопки aButton с методом self.OnClick. Метод Bind() – это метод класса wx.EvtHandler, который является базовым классом всех отображаемых объектов. Поэтому, этот пример кода может быть применен к любому виджету.

Вам кажется, что программа wxPython пассивно ждет событие, но на самом деле она выполняет метод wx.App.MainLoop(). MainLoop() может быть переведен в упрощенный псевдокод Python:

```
while True:
    while not self.Pending():
        self.ProcessIdle()
    self.DoMessage()
```

Другими словами, пока нет событий, система простаивает. При появлении события, посылается сообщение соответствующему обработчику.

3.2.2 Проектирование программ управляемых событиями

Природа управляемой событиями программы wxPython предполагает определенные способы проектирования и кодирования. Так как неизвестно, когда произойдет событие, программист уступает большую часть управления программой пользователю. Большая часть кода в вашей программе wxPython выполняется как прямой или косвенный результат действий пользователя или системы. Например, сохранение документа в вашей программе происходит после того, как пользователь выберет пункт меню, нажмет кнопку панели инструментов или нажмет горячую клавишу. Любое из этих событий может вызвать обработчик, который сохраняет документ пользователя.

Архитектура программ, управляемых событиями, является распределенной. Код, который вызывается в ответ на событие, обычно не определяется виджетом, который вызвал это событие. Например, программный код, вызванный в ответ на нажатие кнопки, не обязан быть частью определения кнопки. Он может быть определен во фрейме или любом другом месте. Объединение такой архитектуры с объектно-ориентированным дизайном позволяет создавать программный код многократного использования. Вы найдете, что гибкая природа Python делает особенно простым повторное использование общих обработчиков событий в различных приложениях wxPython. С другой стороны, распределенная природа программы, управляемой событиями, затрудняет ее понимание и поддержку. Иногда бывает трудно разыскать метод, вызываемый в ответ на событие. (В некоторой степени, эта проблема верна для всего объектно-ориентированного программирования). В главе 5 мы дадим рекомендации, позволяющие упорядочить код программ, управляемых событиями.

3.2.3 Генерация событий

В wxPython, большинство виджетов генерирует высокоуровневые события в ответ на события более низкого уровня. Например, щелчок мыши на кнопке wx.Button генерирует событие wx.CommandEvent типа EVT_BUTTON. Точно так же при перетаскивании мышью угла окна, wxPython автоматически создаст событие wx.SizeEvent. Преимущество высокоуровневых событий состоит в том, что они позволяют сосредоточиться на самих событиях, вместо того, чтобы отслеживать каждый щелчок мыши. Высокоуровневые события могут также содержать полезную информацию о событии. Поскольку вы создаете ваши собственные виджеты, вы можете определить ваши собственные события.

В wxPython события – это объекты класса wx.Event или производного от него. Базовый класс wx.Event – это небольшой абстрактный класс, содержащий методы чтения и записи для свойств, определенных для всех событий, типа EventType, EventObject, и Timestamp. Различные подклассы wx.Event добавляют дополнительную информацию. Например, wx.MouseEvent содержит информацию о координатах курсора миши и нажатой кнопке.

В wxPython определено несколько различных подклассов wx.Event. Таблица 3.2 содержит список некоторых из классов событий, с которыми вы скорее всего столкнетесь. Помните, один класс события может иметь много типов. Каждый тип события соответствует различным действиям пользователя.

Таблица 3.2 Важнейшие подклассы wx.Event

Событие	**Описание**
wx.CloseEvent	Происходит при закрытии фрейма. Тип события позволяет различить нормальное закрытие фрейма и системное завершение.
wx.CommandEvent	Происходит при взаимодействии пользователя с различными виджетами, типа щелчка кнопки, выбора пункта меню или радио-

	кнопки. Каждое из этих отдельных действий имеет собственный тип. Многие более сложные виджеты, типа списка или сетки (grid), определяют подклассы wx.CommandEvent. Обработка команд отличается от обработки других событий.
wx.KeyEvent	Происходит при нажатии клавиши на клавиатуре. Типы: key down, key up и key press.
wx.MouseEvent	События мыши. Типы события различают перемещение мыши и щелчок мыши. Есть отдельные типы, в зависимости от нажатой кнопки и для одиночного и двойного щелчка.
wx.PaintEvent	Происходит, когда содержимое окна должно быть перерисовано.
wx.SizeEvent	Происходит при изменении размеров окна. Обычно это приводит к изменению размеров или расположения элементов окна.
wx.TimerEvent	Может быть создано классом wx.Timer, который генерирует периодические события.

Как правило, объекты событий сами ничего не делают. Событие передается как параметр соответствующему обработчику, используя биндер и систему обработки событий.

3.3 Как я связать событие с обработчиком?

Биндеры состоят из объектов класса wx.PyEventBinder. Объекты wx.PyEventBinder определены для всех типов поддерживаемых событий. Вы можете создать ваши собственные биндеры для ваших собственных типов событий, когда это необходимо. Для каждого типа события определен свой биндер. Типы событий более детализированы чем подклассы wx.Event. Например, класс wx.MouseEvent имеет четырнадцать отдельных типов событий.

В wxPython, имена объектов биндеров глобальны. Чтобы ясно связать типы объектов с обработчиками, эти имена начинаются с wx.EVT_ и соответствуют названиям макроопределений, используемых в коде C++ wxWidgets. В коде wxPython, имя биндера используются вместо типа события. В результате, и это стоит подчеркнуть, имя биндера – это не целочисленный код, который вы получили бы, вызывая метод GetEventType() объекта wx.Event. Челочисленные коды типов событий имеют полностью различный набор глобальных названий, и не часто используются практически.

Для примера рассмотрим типы событий wx.MouseEvent. Как мы упоминали, их – четырнадцать. Девять из них описывают нажатия кнопок мыши. Вот их имена:

```
wx.EVT_LEFT_DOWN
wx.EVT_LEFT_UP
wx.EVT_LEFT_DCLICK
wx.EVT_MIDDLE_DOWN
wx.EVT_MIDDLE_UP
wx.EVT_MIDDLE_DCLICK
wx.EVT_RIGHT_DOWN
wx.EVT_RIGHT_UP
wx.EVT_RIGHT_DCLICK
```

Дополнительно, событие типа wx.EVT_MOTION происходит при перемещении мыши. События типов wx.ENTER_WINDOW и wx.LEAVE_WINDOW возникают, когда курсор мыши входит в область виджета и выходит из нее. Событие типа wx.EVT_MOUSEWHEEL происходит из-за движения колесика мыши. Наконец, вы можете связать все события мыши с единственной функцией, используя тип wx.EVT_MOUSE_EVENTS.

Аналогично, класс wx.CommandEvent имеет 28 различных типов событий, связанных с ним (хотя некоторые из них - только для последних версий Windows). Большинство из них определено для единственного виджета, например, wx.EVT_BUTTON для кнопки и wx.EVT _MENU для выбора пункта меню. События команды описываются вместе со своими виджетами во второй части книги.

Благодаря использованию типов событий, wxPython может очень точно управлять событиями, все еще позволяя подобным событиям быть объектами того же самого класса, и совместно использовать данные и функциональные возможности. Это делает написание обработчиков событий намного более чистыми в wxPython, чем в других инструментариях для создания интерфейса.

Биндеры позволяют связать виджет, событие и функцию обработчик. Эта связь позволяет системе wxPython отвечать на события виджета, выполняя код функции обработчика. В wxPython, любой объект, который может ответить на событие, является подклассом wx.EvtHandler. Все отображаемые объекты – это подклассы wx.EvtHandler, следовательно каждый виджет в приложении wxPython может отвечать на события. Класс wx.EvtHandler может использоваться не только с виджетами, например его использует wx.App. Таким образом объекты, отвечающие на события, не ограничены виджетами. Фраза: виджет может ответить на события, означает, что виджет может создать биндер, который wxPython использует для диспетчеризации событий. Программный код функции обработчика не обязан располагаться в классе wx.EvtHandler.

3.3.1 Работа с методами wx.EvtHandler

Класс wx.EvtHandler определяет множество методов, которые не вызываются при нормальных обстоятельствах. Часто используется только метод Bind(). Он создает биндеры событий, которые мы обсуждали. Метод принимает следующие параметры:

```
Bind(event, handler, source=None, id=wx.ID_ANY, id2=wx.ID_ANY)
```

Функция Bind() связывает событие, объект и обработчик. Параметр event обязательный. Это объект класса wx.PyEventBinder как описано в разделе 3.3. Параметр handler, также обязательный. Это объект, поддерживающий вызов, обычно это метод или функция с единственным параметром – объектом событием. Параметром обработчика может быть None, если событие не связано с обработчиком. Параметр source – это виджет, который является источником события. Параметр используется, когда виджет, вызывающий событие не тот, который используется как обработчик. Как правило, обработчики событий - это методы вашего класса wx.Frame. С этими методами вы связываете события от виджетов, содержащихся в окне. Однако, если родительское окно содержит больше чем один источник события нажатия кнопки (например в окне есть две кнопки OK и Cancel), параметр source используется, чтобы различить какой объект является источником события. Следующий пример демонстрирует это:

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

Связываются событие, объект button (и только button) и метод OnClick(). В листинге 3.1 дан пример использования метода Bind() с параметром source и без него. Вы не обязаны называть ваши обработчики событий On<event>, но это - общее соглашение.

Листинг 3.1 Пример использования метода Bind() с параметром source и без него

```
def __init__(self, parent, id):
    wx.Frame.__init__(self, parent, id, 'Frame With Button',
        size=(300, 100))
    panel = wx.Panel(self, -1)
    button = wx.Button(panel, -1, "Close", pos=(130, 15),
        size=(40, 40))
    # (1) Binding the frame close event
    self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
    # (2) Binding the button event
    self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)

def OnCloseMe(self, event):
```

```
    self.Close(True)

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

(1) В этой строке событие закрытия фрейма связывается с методом self.OnCloseWindow. Так как событие и вызвано и связано с фреймом, нет необходимости передавать параметр source.

(2) Эта строка связывает событие нажатия кнопки с кнопкой и методом self.OnCloseMe. В этом случае, кнопка, генерирующая событие, это не фрейм. Поэтому, кнопку нужно передать как параметр методу Bind, чтобы wxPython мог различать события нажатия этой кнопки и события нажатия других кнопок фрейма.

Вы можете также использовать параметр source, чтобы идентифицировать элемент, даже если элемент не является источником события. Например, вы можете связать события меню с обработчиками даже при том, что события меню технически вызваны фреймом. Листинг 3.2 иллюстрирует пример связывания событий меню.

Листинг 3.2 Связывание событий меню

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

import wx

class MenuEventFrame(wx.Frame):

    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Menus',
                size=(300, 200))
        menuBar = wx.MenuBar()
        menu1 = wx.Menu()
        menuItem = menu1.Append(-1, "&Exit...")
        menuBar.Append(menu1, "&File")
        self.SetMenuBar(menuBar)
        self.Bind(wx.EVT_MENU, self.OnCloseMe, menuItem)

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

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = MenuEventFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

Параметры id и id2 метода Bind() определяют источник события, используя число ID, а не виджет непосредственно. Как правило, id и id2 не требуются, так как идентификатор (ID) источника события может быть извлечен из параметра source. Однако, иногда использование ID непосредственно имеет смысл. Например, если вы используете предопределенные ID для диалогового окна, легче использовать число ID, чем использовать виджет. Если вы используете оба параметра (и id и id2), то вы можете связать с событием несколько виджетов, идентификаторы которых находятся в диапазоне от id до id2. Это полезно, когда идентификаторы виджетов последовательны.

Старый стиль связывания

Метод Bind() появился в wxPython начиная с версии 2.5. В предыдущих версиях wxPython, имена EVT_* использовались как функции, и связывание выглядело следующим образом:

```
wx.EVT_BUTTON(self, self.button.GetId(), self.OnClick)
```

Недостаток старого стиля в том, что он не выглядит и не действует как объектно-ориентированный метод. Однако, старый стиль все еще работает в версии 2.5 (потому что объекты wx.EVT * поддерживают вызов), и вы, возможно, будете встречать его в коде wxPython.

В таблице 3.3 представлен список некоторых методов класса wx.EvtHandler, которые вы можете использовать для управления процессом обработки событий.

Таблица 3.3 Обычно используемые методы класса wx.EvtHandler

Метод	**Описание**
AddPendingEvent(event)	Помещает параметр event в систему обработки событий. Подобен методу ProcessEvent(), но, в отличие от него, не вызывает непосредственную обработку события. Вместо этого событие добавляется в очередь событий.
Bind(event, handler, source=None, id=wx.ID_ANY, id2=wx.ID_ANY)	См. полное описание в разделе 3.3.1.
GetEvtHandlerEnabled() SetEvtHandlerEnabled(boolean)	Свойство имеет значение True, если обработчик в настоящее время обрабатывает события, иначе - False.
ProcessEvent(event)	Помещает объект event в систему обработки событий для непосредственной обработки.

3.4 Как wxPython обрабатывает события?

Ключевой компонент системы, основанной на событиях, - это процесс диспетчеризации событий. В этом разделе, мы рассмотрим все шаги обработки поступающих событий. Для этого мы используем небольшой пример, чтобы проследить шаги в процессе. На рисунке 3.2 изображено окно с единственной кнопкой, которая будет использоваться для генерации событий.

Рисунок 3.2 Окно для тестирования событий

Листинг 3.3 содержит код, который создает это окно. В этом коде, события генерируются при щелчке на кнопке и при перемещении курсора мыши над кнопкой.

Листинг 3.3 Различные виды событий мыши

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

import wx

class MouseEventFrame(wx.Frame):

    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Frame With Button',
                size=(300, 100))
        self.panel = wx.Panel(self)
        self.button = wx.Button(self.panel,
                label="Not Over", pos=(100, 15))
        # (1) Binding the button event
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick,
                self.button)
        # (2) Binding the mouse enter event
        self.button.Bind(wx.EVT_ENTER_WINDOW,
```

```
                self.OnEnterWindow)
        # (3) Binding the mouse leave event
        self.button.Bind(wx.EVT_LEAVE_WINDOW,
                self.OnLeaveWindow)

    def OnButtonClick(self, event):
        self.panel.SetBackgroundColour('Green')
        self.panel.Refresh()

    def OnEnterWindow(self, event):
        self.button.SetLabel("Over Me!")
        event.Skip()

    def OnLeaveWindow(self, event):
        self.button.SetLabel("Not Over")
        event.Skip()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = MouseEventFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

Фрейм MouseEventFrame содержит одну кнопку в середине. Нажатие на мышь изменяет цвет фона фрейма на зеленый. Клик мышкой связывается с обработчиком в строке (1). Когда курсор мыши входит в область кнопки, измененяется заголовок кнопки, связь с обработчиком в строке (2). Когда курсор мыши покидает кнопку, заголовок меняется на прежний, связь с обработчиком в строке (3).

При рассмотрении этого примера, возникают некоторые вопросы об обработке событий в wxPython. В строке (1), событие кнопки, генерируется кнопкой, расположенной на фрейме. Откуда wxPython знает, что искать биндер нужно в объекте frame, а не объекте button? В строках (2) и (3), события связываются непосредственно с объектом button. Почему эти события не могут также быть связаны с фреймом? На оба этих вопроса мы ответим после исследования процедуры обработки событий wxPython.

3.4.1 Понимание процесса обработки событий

Процедура обработки событий в wxPython была спроектирована так, чтобы сделать ее простой для программиста, чтобы обрабатывать высокоуровневые события и игнорировать низкоуровневые. Далее, мы проследим процедуру обработки событий для нажатия кнопки и щелчка мышью.

На рисунке 3.3 изображена основная блок-схема процесса обработки событий. Прямоугольники указывают начало и конец процесса, круги указывают различные объекты wxPython, которые являются частью процесса, ромбы указывают пункты решения, и прямоугольники с полосами указывают фактические методы обработчики событий.

Процесс начинается с объекта, который вызвал событие. Как правило, wxPython ищет сначала у этого объекта обработчик соответствующего типа события. Если обработчик найден, то он выполняется. В противном случае wxPython проверяет, распространять ли событие по контейнерной иерархии. Если да, то проверяется родительский контейнер и все контейнеры до окна верхнего уровня. Если событие не распространяется, то wxPython проверяет прикладной объект (wx.App) для поиска метода обработчика. После выполнения обработчика процесс обычно заканчивается. Однако, обработчик может сказать wxPython продолжать искать обработчики.

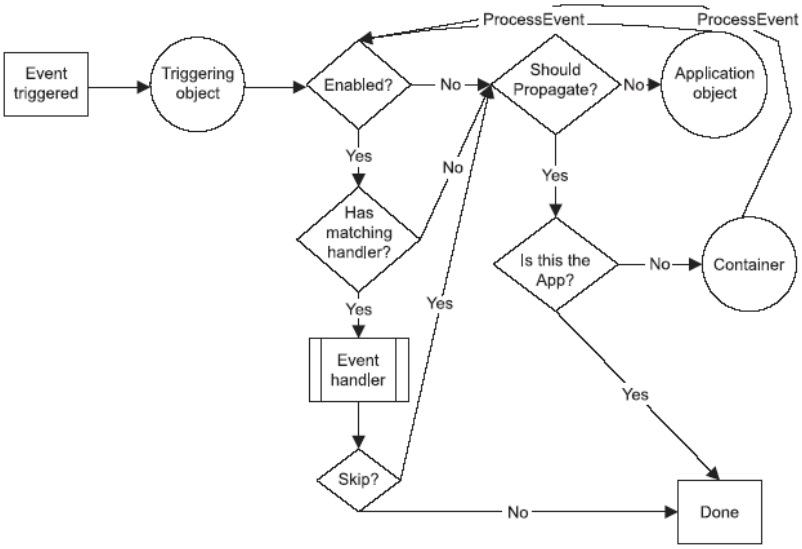

Рисунок 3.3

Давайте подробно рассмотрим каждый шаг процесса. Перед обсуждением каждого шага, мы отобразим эскиз для соответствующего фрагмента рисунка 3.3.

Шаг 1 Создание события

Процесс начинается, когда событие произошло.

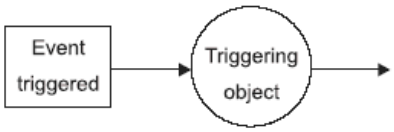

Рисунок 3.4

Большинство существующих типов событий создается в ответ на определенные пользовательские действия или системные уведомления. Например, событие mouse entering создается, когда wxPython замечает вход мыши в границы нового виджета, и событие button click создается после событий left mouse down и left mouse up на кнопке.

Событие сначала передается объекту, ответственному за создание события. Для button click, объект - кнопка, для mouse enter event, объект – виджет, в который входит курсор мыши.

Шаг 2 Определяет, разрешено ли объекту обрабатывать события

Следующий шаг процесса проверяет, может ли виджет ответственный за событие обработать это события.

Окну можно разрешить или запретить обработку события, вызывав метод wx.EvtHandler SetEvtHandlerEnabled(boolean). Эффект запрещения обработки события состоит в том, что виджет полностью исключается из процесса обработки события.

Разрешение (enable) или запершение (disable) виджета на уровне обработки события не нужно путать с запрещением виджета на уровне пользовательского интерфейса (UI). Отключение виджета на уровне UI можно выполнить, используя методы Disable() и Enable() класса wx.Window. Отключение виджета в UI означает, что пользователь не может взаимодействовать с заблокированным виджетом. Обычно, заблокированный виджет показан на экране серым цветом, чтобы указать на его состояние. Окно, которое было заблокировано на уровне UI, не будет в состоянии произвести любые события; однако, если оно находится на контейнерной иерархии для других событий, оно все еще обрабатывает события, которые

получает. До конца этого раздела, мы будем использовать термины enabled и disabled имея в виду уровень обработки события.

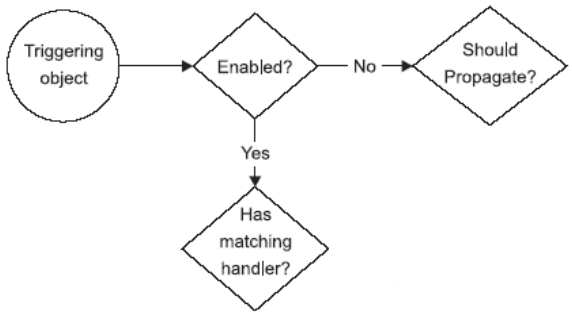

Рисунок 3.5

Проверка состояния объекта enabled/disabled выполняется в методе ProcessEvent(), который вызывает система wxPython, чтобы запустить механизм диспетчеризации событий. Мы будем встречать метод ProcessEvent() снова и снова при рассмотреднии процесса обработки событий. Он фактически осуществляет большую часть процесса, изображенного на рисунке 3.3. Метод ProcessEvent() возвращает True, если обработка события выполнена. Обработку считают выполненой, если найдена функция обработчика для объекта и обрабатываемого события. Функция обработчика может явно просить продолжить обработку, вызвав метод Skip() класса wx.Event. Кроме того, если объект - подкласс wx.Window, он может фильтровать события, используя специальный объект - validator. Validators рассматриваются более подробно в главе 9.

Шаг 3 Определение расположения обработчика

Метод ProcessEvent() ищет объект binder, который связывает тип события и текущий объект.

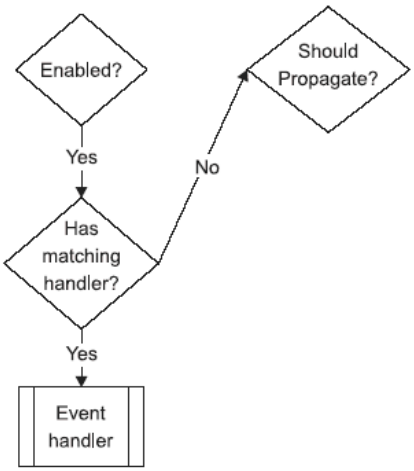

Рисунок 3.6

Если binder не найден для объекта непосредственно, обработка идет по иерархии класса, чтобы найти binder в родительском классе объекта — это отличается от обхода в контейнерной иерархии, которая выполняется в следующем шаге. Если объект binder найден, wxPython вызывает связанную функцию обработчика. После того, как обработчик вызван, обработка этого события завершается, если только функция обработчика явно не запрашивает дальнейшую обработку.

В листинге 3.3 событие mouse enter перехватывается, потому что определена связь между кнопкой, биндером wx.EVT_ENTER_WINDOW и методом OnEnterWindow(), который и вызывается. Так как мы не связывали событие mouse button click и wx.EVT_LEFT_DOWN, wxPython продолжил бы поиск в этом случае.

Шаг 4 Определяет продолжать ли обрабатывать событие

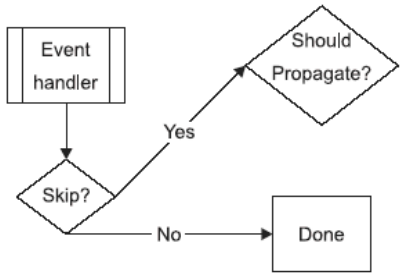

Рисунок 3.7

После вызова первого обработчика, wxPython проверяет требуется ли дальнейшая обработка. Обработчик запрашивает дальнейшую обработку, вызывая метод Skip() класса wx.Event. Если метод Skip() вызывается, то обработка продолжается, и любые обработчики, определенные в суперклассе выполняются в этом шаге. Метод Skip() можно вызвать в любом месте обработчика, или любом коде, вызванном обработчиком. Метод Skip() устанавливает в объекте события флаг, который wxPython проверяет после выполнения метода обработчика. В листинге 3.3 OnButtonClick() не вызывает Skip(), таким образом процесс обработки события завершается в конце метода обработчика. Другие два обработчика события вызывают Skip(), поэтому система продолжит искать соответствующий биндер для события, в конечном счете вызывая функцию обработки события по умолчанию.

Шаг 5 Определяет распространять ли событие

В конечном счете wxPython определяет, должен ли процесс распространять событие по контейнерной иерархии, чтобы найти обработчик. Контейнерная иерархия – это путь от определенного виджета до фрейма верхнего уровня, проходящий от каждого виджета вверх к его родительскому контейнеру.

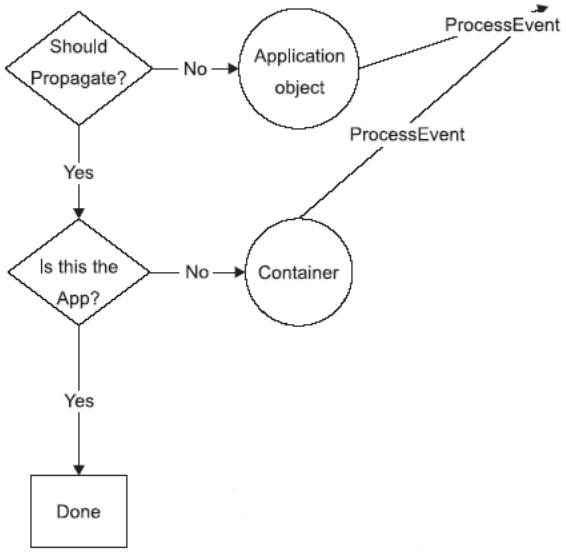

Рисунок 3.8

Если текущий объект не имеет обработчика для события, или если обработчик вызывает метод Skip(), wxPython определяет, должено ли событие распростанятся по контейнерной иерархии. Если ответ «Нет», процесс еще раз ищет обработчик, в объекте wx.App, и затем останавливается. Если ответ «Да», процесс переходит к контейнеру виджета. Процесс продолжается пока wxPython не находит соответствующий биндер, или не достигает фрейма верхнего уровня, или объекта wx.Dialog (даже если диалог не окно верхнего уровня). Считают, что событие нашло соответствующий биндер, если ProcessEvent () для этого объекта возвращает True, указывая, что обработка завершена.

Должено ли событие распространятся по контейнерной иерархии определяет динамическое свойство каждого объекта события, хотя на практике почти всегда используется значение по умолчанию. По умолчанию, только объекты класса wx.CommandEvent и производные от него, распространяются по контейнерной иерархии. Все другие события этого не делают.

В листинге 3.3 так обрабатывается нажатие кнопки. Щелчок мышью на wx.Button генерирует событие command типа wx.EVT_BUTTON. После того, как wxPython не в состоянии найти биндер в объекте кнопки, он передает событие родителю, который в этом случае является панелью. У панели нет соответствующего биндера, далее проверяется родитель панели - фрейм. Так как фрейм имеет соответствующий биндер, ProcessEvent() вызывает соответствующую функцию - OnButtonClick().

Шаг 5 также объясняет, почему события mouse enter и mouse leave должны быть связаны с кнопкой а не с фреймом. Так как события мыши не подкласс wx.CommandEvent, события mouse enter и mouse leave не распространяются вверх родителю, и wxPython не может найти соответствующий биндер.

Событиям command даются такие привилегии, потому что это высокоуровневые события, указывающие, что пользователь делает что-то в приложении, а не в конкретном окне. Предполагается, что события окна прежде всего представляют интерес для виджета, который первоначально их получает, в то время как события прикладного уровня могут представлять интерес выше в иерархии. Это правило не препятствует нам определять обработчик события в другом месте. Например, даже при том, что событие mouse click связано с кнопкой, сам биндер определен в классе фрейма, и вызывает метод класса фрейма. Другими словами, события низкого уровня non-command обычно используются для вещей, которые случаются с виджетом типа щелчка мыши, нажатия клавиши, запроса на перерисовку, изменение размера или перемещение. С другой стороны, события command, типа нажатия кнопки или выбора из списка, обычно производятся и испускаются непосредственно виджетом. Например, события команды кнопки генерируются после нажатия и отпускания кнопки мыши на соответствующем виджете.

Наконец, если событие не обработано после прохождения через контейнерную иерархию, ProcessEvent() вызывается для прикладного объекта wx.App. По умолчанию он ничего не делает, однако, вы можете добавить биндеры событий к вашему wx.App, чтобы обработать события некоторым нестандартным способом. Например, если вы пишите систему для создания GUI, вы можете захотеть, чтобы события в вашем окне построителя интерфейса распространялись и в окно редактирования кода, даже при том, что это оба окна верхнего уровня. Один из способов это сделать - фиксировать события в прикладном объекте и передавать их в окно кода.

3.4.2 Использование метода Skip()

Первый обработчик, найденный для события останавливает процесс обработки события, если только в обработчике не вызван метод Skip(). Метод Skip() позволяет продолжить поиск обработчиков в родительских классах и родительских контейнерах. В некоторых случаях, объект должен продолжить обрабатывать событие, чтобы позволить поведение по умолчанию наряду с вашим собственным обработчиком. В листинге 3.4 показан пример использования метода Skip(), который позволяет программе отвечать и на событие left button down и на событие button click в той же самой кнопке.

Листинг 3.4 Обработка mouse down и button click в то же самое время

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

import wx

class DoubleEventFrame(wx.Frame):
```

```
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Frame With Button',
                size=(300, 100))
        self.panel = wx.Panel(self, -1)
        self.button = wx.Button(self.panel, -1, "Click Me", pos=(100, 15))
        # (1)
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, self.button)
        # (2)
        self.button.Bind(wx.EVT_LEFT_DOWN, self.OnMouseDown)

    def OnButtonClick(self, event):
        self.panel.SetBackgroundColour('Green')
        self.panel.Refresh()

    def OnMouseDown(self, event):
        self.button.SetLabel("Again!")
        # (3)
        event.Skip()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = DoubleEventFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

(1) Эта строка связывает кнопку, событие button click и обработчик OnButtonClick(), который изменяет цвет фона фрейма.

(2) Эта строка связывает событие left mouse button down и обработчик OnMouseDown(), который изменяет текст кнопки. Так как событие left mouse button down не событие command, это событие должно быть связано с кнопкой, а не с фреймом.

Когда пользователь щелкает мышью по кнопке, сначала система генерирует событие left mouse button down. При нормальных обстоятельствах, событие left mouse button down изменяет состояние кнопки таким образом, что последующее событие left button up, создает событие wx.EVT_BUTTON. Класс DoubleEventFrame сохраняет это поведение благодаря вызову метода Skip() в строке (3). Без Skip() алгоритм обработки события находил бы первым биндер, созданный в строке (2), и останавливался прежде, чем кнопка могла создать событие wx.EVT_BUTTON. С использованием метода Skip(), обработка события продолжается, и button click создается.

Связывая свои обработчики с низкоуровневыми событиями, такими как mouse up/down, помните, что wxPython тоже ожидает эти события, чтобы обработать или сгенерировать дальнейшие высокоуровневые события. Если вы не вызываете метод Skip(), то вы рискуете блокировать ожидаемое поведение. Например, вы можете не увидеть как нажимается кнопка.

3.5 Методы wx.App для управления событиями

Вы можете непосредственно управлять главным циклом обработки событий, используя некоторые методы wx.App. Например, вы можете захотеть начать обрабатывать следующее доступное событие в вашем собственном списке, вместо того, чтобы ждать пока wxPython начнет процесс обработки. Эта особенность необходима, если вы запускаете долгую процедуру, и не хотите, чтобы казалось, что GUI застыл. Вы не должны часто использовать методы из этого раздела, но иногда важно иметь такие возможности.

В таблице 3.4 содержится список методов wx.App, которые вы можете использовать для воздействия на цикл обработки событий.

Таблица 3.4 Методы wx.App для цикла обработки событий

Метод	**Описание**

Dispatch()	Диспетчеризирует следующее событие из очереди событий. Используется методом MainLoop(), или в собственных циклах обработки событий.
Pending()	Возвращает True, если в очереди есть события, ожидающие обработки.
Yield(onlyIfNeeded=False)	Позволяет ожидать wxWidgets события, которые будут посланы в середине длительного процесса, который иначе мог бы блокировать систему работы с окнами. Возвращает True, если ожидаются события для обработки, иначе возвращает False. Если True, параметр onlyIfNeeded вынуждает процесс уступать, если в очереди есть события для обработки. Если параметр - False, то это - ошибка вызвать Yield рекурсивно (?). Есть также глобальная функция wx.SafeYield (), которая препятствует пользователю вводить данные в течение выполнения задачи (временно отключая пользовательские виджеты ввода данных). Это препятствует пользователю делать что-то, что нарушило бы состояние выполняющейся задачи.

Другой метод для управления событиями состоит в том, чтобы создать свои собственные типы событий, которые соответствуют специфическим особенностям вашего приложения и виджетов. В следующем разделе, мы обсудим, как создать собственные события.

3.6 Как создать собственные события?

Хотя это большая тема, этот раздел - самое очевидное место, чтобы рассмотреть создание собственных событий. При первом чтении, вы можете пропустить этот раздел, и вернуться к нему позже. В дополнение к различным классам событий, поставляемым wxPython, вы можете создать ваши собственные события. Вы можете создавать их в ответ на обновление данных или другие изменения, которые являются специфическими для вашего приложения, и объекты события должны содержать ваши дополнительные данные. Другая причина создавать собственны класс событий может состоять в том, чтобы поддержать собственный виджет. В следующем разделе, мы рассмотрим пример создания собственного виджета.

3.6.1 Определение своего события для своего виджета

На рисунке 3.9 изображен виджет - панель, содержащая две кнопки. Пользовательское событие TwoButtonEvent вызывается только после того, как пользователь щелкнул обеими кнопками. Событие содержит количество нажатий кнопок. Здесь показывается, как новое событие command может быть создано из низкоуровневых событий – в данном случае, из событий left button down на каждой кнопке.

Рисунок 3.9

Для создания собственного события выполните следующие шаги:

1. Определите новый класс, производный от wx.PyEvent. Если вы хотите, чтобы событие обрабатывалось как команда, то создайте событие как подкласс wx.PyCommandEvent.
2. Создайте тип события и биндер, чтобы связать событие с определенными объектами.
3. Добавьте код, который позволяет создавать объекты нового класса событий, и вводит эти объекты в систему обработки событий, используя метод ProcessEvent(). Как

только событие создано, вы можете связывать его с обработчиками как любые другие события wxPython. В листинге 3.5 показан код для управления виджетом.

Листинг 3.5 Построение собственного виджета с двумя кнопками

```
import wx

class TwoButtonEvent(wx.PyCommandEvent):    # (1)
    def __init__(self, evtType, id):
        wx.PyCommandEvent.__init__(self, evtType, id)
        self.clickCount = 0

    def GetClickCount(self):
        return self.clickCount

    def SetClickCount(self, count):
        self.clickCount = count

myEVT_TWO_BUTTON = wx.NewEventType()    # (2)
EVT_TWO_BUTTON = wx.PyEventBinder(myEVT_TWO_BUTTON, 1)    # (3)

class TwoButtonPanel(wx.Panel):
    def __init__(self, parent, id=-1, leftText="Left",
            rightText="Right"):
        wx.Panel.__init__(self, parent, id)
        self.leftButton = wx.Button(self, label=leftText)
        self.rightButton = wx.Button(self, label=rightText,
                                     pos=(100,0))
        self.leftClick = False
        self.rightClick = False
        self.clickCount = 0
        self.leftButton.Bind(wx.EVT_LEFT_DOWN, self.OnLeftClick)    # (4)
        self.rightButton.Bind(wx.EVT_LEFT_DOWN, self.OnRightClick)

    def OnLeftClick(self, event):
        self.leftClick = True
        self.OnClick()
        event.Skip()    # (5)

    def OnRightClick(self, event):
        self.rightClick = True
        self.OnClick()
        event.Skip()    # (6)

    def OnClick(self):
        self.clickCount += 1
        if self.leftClick and self.rightClick:
            self.leftClick = False
            self.rightClick = False
            evt = TwoButtonEvent(myEVT_TWO_BUTTON, self.GetId())    # (7)
            evt.SetClickCount(self.clickCount)
            self.GetEventHandler().ProcessEvent(evt)    # (8)


class CustomEventFrame(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Click Count: 0',
                          size=(300, 100))
        panel = TwoButtonPanel(self)
        self.Bind(EVT_TWO_BUTTON, self.OnTwoClick, panel)    # (9)

    def OnTwoClick(self, event):    # (10)
        self.SetTitle("Click Count: %s" % event.GetClickCount())

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = CustomEventFrame(parent=None, id=-1)
    frame.Show()
```

```
app.MainLoop()
```

(1) Класс для события обьявляется как подкласс wx.PyCommandEvent. Для создания новых классов событий нужно использовать классы wx.PyEvent и wx.PyCommandEvent, чтобы создать прослойку между классами C++ и вашим кодом Python. Если вы попробуете использовать wx.Event непосредственно, wxPython не сможет увидеть новые методы вашего подкласса при обработке события, потому что обработчики событий C++ не знают о подклассе Python. Если Вы используете wx.PyEvent, ссылки на события Python сохраняются и позже передаются обработчикам непосредственно, позволяя использоваться части Python кода.

(2) Глобальная функция wx.NewEventType() похожа на wx.NewId(). Она возвращает уникальный идентификатор для типа события. Это значение уникально идентифицирует тип события для системы обработки событий.

(3) Объект binder создается, используя новый тип события как параметр. Второй параметр – число из диапазона от 0 до 2, представляющее количество идентификаторов wxId, ожидаемых методом wx.EvtHandler.Bind(), чтобы определить, какой объект является источником события. В нашем случае, есть один ID, представляющий виджет, который генерирует событие.

(4) Чтобы создать новое высокоуровневое событие типа command, программа должна ответить на определенные низкоуровневые события, например, нажатие мыши на каждой кнопке. В зависимости от того, какая кнопка нажата, события связываются с методами OnLeftClick() и OnRightClick(). Обработчики устанавливают логические значения, указывая, что кнопка была нажата.

(5) (6) Вызов Skip() заставляет продолжить обработку события. Обычно, все низкоуровневые события должны вызывать Skip(), чтобы не блокировать возможную обработку по умолчанию.

Мы не использовали связь с событием wx.EVT_BUTTON, чтобы показать вам, что случится, если не вызвать Skip(). Чтобы увидеть различие в поведении, закомментируйте строку (5) или (6).

(7) Если и левая и правая кнопки нажаты, создается новое событие. Тип события и идентификатор виджета - параметры конструктора события. Как правило, единственный класс события может иметь больше чем один тип события, хотя в этом примере тип единственный.

(8) Вызов ProcessEvent() вводит новое событие в систему обработки событий, как описано в разделе 3.4.1. Вызов GetEventHandler() возвращает объект wx.EvtHandler. В большинстве случаев, возвращаемый объект – это объект виджета.

(9) Собственное событие связывается точно так же как любое другое событие, в этом случае используя объект binder, созданный в строке (3).

(10) Обработчик события в этом примере изменяет заголовок окна, чтобы отобразить новое количество щелчков.

Создание событий - важная часть настройки wxPython.

3.*7 Резюме*

- Приложение wxPython управляется событиями. Большую часть времени приложение проводит в главном цикле, ожидая события и посылая их соответствующим обработчикам.

- Все события wxPython –классы производные от wx.Event. Низкоуровневые события типа щелчка мыши, используются, чтобы создать события более высокого уровня типа нажатия кнопки или выбора пункта меню. Эти высокоуровневые события - подклассы класса wx.CommandEvent. Большинство классов событий имеют разделение на категории по типам событий, которые могут использовать тот же самый набор параметров.

- Чтобы определить связьи между событиями и функциями, wxPython использует объекты класса wx.PyEventBinder. Есть много предопределенных объектов этого класса, каждый объект соответствует определенному типу события. Каждый виджет wxPython - подкласс wx.EvtHandler. Класс wx.EvtHandler имеет метод Bind(), который обычно вызывается при инициализации объекта с параметрами event binder и функцией обработчиком. В зависимости от типа события, метод Bind() может потребовать дополнительные параметры.

- События сначала посылают объекту, который произвел их, чтобы найти биндер, который связан с обработчиком. Если событие - команда, событие распространяется вверх по контейнерной иерархии, пока не найдется виджет, который имеет обработчик для этого типа события. Как только обработчик найден, он выполняется, и на этом обработка события останавливается, если только обработчик не вызвал метод Skip(). Вы можете использовать метод Skip(), чтобы позволить вызов нескольких обработчиков для обработки единственного события, или заставить выполнить обработку по умолчанию. Определенными аспектами главного цикла можно управлять, используя методы wx.App.

- В wxPython могут быть созданы собственные события, которые будут генерироваться как часть поведения собственного виджета. Собственные события - подклассы wx.PyEvent, собственные события команды - подклассы wx.PyCommandEvent. Чтобы создать собственное событие, должен быть определен новый класс, и должен быть создан объект binder для каждого типа события нового класса. Наконец, событие должно быть сгенерировано где-нибудь в системе, используя метод ProcessEvent().

В этой главе, мы рассмотрели важные объекты вашего приложения wxPython. В следующей главе, мы покажем вам полезный инструмент, который поможет вам разрабатывать приложения wxPython.