Часть 2

Основные элементы wxPython

В этой части книги мы изучим важнейшие виджеты, которые составляют ядро wxPython. Эти базовые элементы будут важной частью любой создаваемой Вами wxPython-программы. Для каждого элемента, мы покажем Вам наиболее важные части API для работы с этим элементом, а также образец кода и рекомендации того, как использовать этот элемент в реальных программах.

Глава 7, «Работа с основными элементами управления», знакомит с основным набором виджетов. Мы рассмотрим текстовые метки, ввод текста, кнопки, а также виджеты для числового и списочного выбора. Мы покажем Вам, как использовать каждый элемент, как модифицировать его вид, чтобы он соответствовал Вашему приложению, и как реагировать на действия пользователя. В главе 8, «Размещение виджетов на фрейме», мы пройдемся по иерархии контейнеров и поговорим о фреймах. Мы покажем Вам, как добавлять виджеты во фрейм и опишем доступные фреймовые стили. Мы также обсудим жизненный цикл фрейма от его создания до уничтожения. В главе 9, «Работа пользователей с диалогами», мы сосредоточимся на диалогах, начиная обсуждение с того, чем диалоговые контейнеры отличаются от фреймов. Мы также покажем доступный в wxPython ряд встроенных диалогов, а также удобные способы их использования.

Глава 10, «Создание и использование меню wxPython», посвящена меню. Мы обсудим, как создавать подключаемые к меню и размещаемые в строке меню пункты меню. Мы также рассмотрим меню в виде переключателей, всплывающие меню (pop-up) и различные способы настройки отображения Вашего меню. В главе 11, «Размещение виджетов с помощью координаторов (sizer)», мы раскроем технологию координаторов. Координаторы используются для упрощения компоновки виджетов на фреймах и диалогах wxPython. Мы изучим шесть типов встроенных координаторов, покажем, как они себя ведут, и дадим некоторые рекомендации по тому, когда их лучше всего использовать. Наконец, в главе 12, «Основы манипулирования графическими изображениями», мы обсудим элементарные основы рисования на экране при помощи контекста устройства. Здесь перечисляются методы рисования примитивов, которые Вы можете использовать для рисования своих собственных виджетов, для поддержки функций рисования пользователя, или просто для украшения.

7 Работа с основными элементами управления

Эта глава включает

- Вывод текста
- Работа с кнопками
- Ввод и отображение чисел
- Предоставление пользователю возможности выбора

Инструментарий wxPython предоставляет множество различных виджетов, включая базовые элементы управления, являющиеся темой этой главы. Мы исследуем начальный набор виджетов, содержащий следующие элементы управления: статический текст, редактируемый текст, кнопки, счетчики (spinners), ползунки (slider), флажки (check box), переключатели (radio button), списки (list box), выпадающие списки (choice), комбинированные списки (combo box) и индикаторы прогресса (gauge). Для каждого виджета мы покажем короткий пример его использования, сопровождаемый описанием важных частей API wxPython.

7.1 Вывод текста

Этот раздел начинается с примеров отображения текста на экране, которые содержат используемые Вами в качестве меток стилизованные и простые поля статического текста. Вы также сможете создать текстовые поля для однострочного или многострочного ввода пользователя. К тому же, мы рассмотрим, как выбрать шрифт текста.

7.1.1 Как вывести статический текст?

Возможно, основная задача любого пакета для разработки интерфейса пользователя состоит в рисовании на экране простого текста. В wxPython это выполняется на базе класса `wx.StaticText`. Рисунок 7.1 отображает статические текстовые элементы.

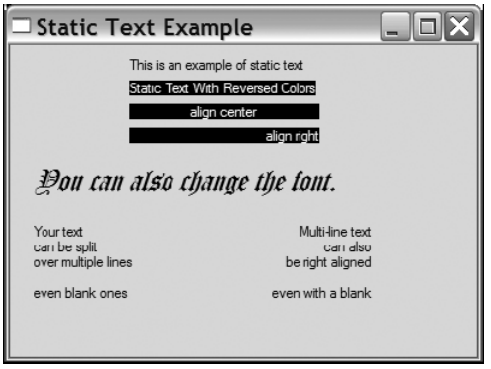

Рисунок 7.1 Примеры виджетов wx.StaticText с изменением шрифта, выравнивания и цвета

В `wx.StaticText` Вы можете изменять выравнивание, шрифт и цвет текста. Отдельный виджет статического текста может содержать многострочный текст, однако, он не может управлять множеством шрифтов или стилей. Для поддержки множества шрифтов или стилей используйте более сложный текстовый элемент управления, например, `wx.html.HTMLWindow`, описываемый в главе 16. Для того, чтобы отображать множество строк внутри статического текстового поля, включите в него строку с символами новой строки и сделайте элемент достаточно большим, чтобы отображать весь текст. Единственная особенность, которая не видна на рисунке, состоит в том, что окно `wx.StaticText` никогда не получает и не реагирует на события мыши, и оно никогда не получает фокус ввода.

Вывод статического текста

Листинг 7.1 показывает текст программы, соответствующий рисунку 7.1

Листинг 7.1 Базовый пример использования статического текста

```
import wx

class StaticTextFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Static Text Example',
                size=(400, 300))
        panel = wx.Panel(self, -1)
        wx.StaticText(panel, -1, "This is an example of static text",
                (100, 10))
        rev = wx.StaticText(panel, -1,
                "Static Text With Reversed Colors",
                (100, 30))
        rev.SetForegroundColour('white')
        rev.SetBackgroundColour('black')
        center = wx.StaticText(panel, -1,
                "align center", (100, 50),
                (160, -1), wx.ALIGN_CENTER)
        center.SetForegroundColour('white')
        center.SetBackgroundColour('black')
        right = wx.StaticText(panel, -1,
                "align right", (100, 70),
                (160, -1), wx.ALIGN_RIGHT)
        right.SetForegroundColour('white')
        right.SetBackgroundColour('black')
        str = "You can also change the font."
        text = wx.StaticText(panel, -1, str, (20, 100))
        font = wx.Font(18, wx.DECORATIVE,
               wx.ITALIC, wx.NORMAL)
        text.SetFont(font)
        wx.StaticText(panel, -1,
                "Your text\ncan be split\n"
                "over multiple lines\n\neven blank ones", (20,150))
        wx.StaticText(panel, -1,
                "Multi-line text\ncan also\n"
                "be right aligned\n\neven with a blank", (220,150),
                style=wx.ALIGN_RIGHT)

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

Конструктор для `wx.StaticText` идентичен базовым конструкторам wxWidget:

```
wx.StaticText(parent, id, label, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=0, name="staticText")
```

Таблица 7.1 показывает назначение его параметров - подобный набор параметров в своих конструкторах имеет большинство виджетов wxPython. За детальным описанием параметров конструктора обращайтесь к обсуждению виджетов в главе 2.

Таблица 7.1 Параметры конструктора `wx.StaticText`

Параметр	Назначение
`parent`	Виджет, вмещающий данный статический текст
`id`	Идентификатор wxPython. Для автоматического создания уникального идентификатора необходимо использовать значение -1.
`label`	Текст, который необходимо отобразить
`pos`	Позиция виджета на базе объекта `wx.Point` или кортеж (tuple) Python
`size`	Размер создаваемого виджета на базе объекта `wx.Size` или кортеж (tuple) Python
`style`	Флаг стиля
`name`	Имя объекта, используемое при поиске

В следующем разделе мы подробно обсудим стилевые флаги.

Работа со стилями

Все методы, вызываемые для экземпляров статического текста в листинге 7.1, принадлежат родительскому классу `wx.Window`; `wx.StaticText` не определяет каких-либо новых собственных методов. Несколько характерных для `wx.StaticText` стилевых флагов указаны в таблице 7.2.

Таблица 7.2 Стилевые флаги класса `wx.StaticText`

Стиль	Описание
`wx.ALIGN_CENTER`	Центрирует статический текст внутри размерного прямоугольника виджета
`wx.ALIGN_LEFT`	Выравнивание текста виджета слева. Этот стиль установлен по умолчанию.
`wx.ALIGN_RIGHT`	Выравнивание текста виджета справа.
`wx.ST_NO_AUTORESIZE`	Если использован этот флаг, виджет со статическим текстом не будет самостоятельно изменять свои размеры после изменения текста методом `SetLabel()`. Чтобы сохранять выравнивание, Вам нужно использовать данный стиль вместе с центрированием или выравниванием поля справа.

Элемент управления `wx.StaticText` перекрывает метод `SetLabel()` для изменения своих размеров на основании нового текста, что происходит, если не установлен стиль `wx.ST_NO_AUTORESIZE`.

При создании однострочного поля статического текста с центрированием или выравниванием справа, Вы должны явно устанавливать в конструкторе размер этого поля. Определение размера сдерживает wxPython от автоматического изменения размеров элемента управления. В wxPython размер по умолчанию соответствует минимальному окружающему текст прямоугольнику. Так как по умолчанию размер текстового поля никак не больше размещенного в нем текста и при отображении выравнивания пустого пространства не образуется, не имеет значения выравнено ли поле слева, справа или по центру. Для того чтобы при выполнении программы изменять текст в виджете динамически без изменения размера поля, установите стиль `wx.ST_NO_AUTORESIZE`. Это не даёт виджету после изменения текста привести свои размеры к минимальному прямоугольнику. Если статический текст размещается внутри динамической формы, изменение его размера может переместить на экране другие виджеты, отвлекая при этом внимание пользователя.

Другие приёмы вывода текста

Имеются и другие способы размещения статического текста на Вашем экране. Во-первых, можно использовать класс `wx.lib.stattext.GenStaticText`, который представляет собой видоизмененную Python- реализацию класса `wx.StaticText`. Он более согласован в кросс-платформенном плане, чем стандартная версия C++, и он принимает события мыши. Он также предпочтителен, когда Вам необходим подкласс при создании собственного статического текстового элемента управления.

Вы можете рисовать текст непосредственно в Вашем контексте устройства при помощи метода `DrawText(text, x, y)` или метода `DrawRotatedText(text, x, y, angle)`. Последний является простейшим способом вывести текст под углом, хотя имеется также управляющий вращением подкласс `GenStaticText`. Контексты устройства были кратко рассмотрены в главе 6 и будут рассмотрены более подробно в главе 12.

7.1.2 Как обеспечить пользователя возможностью ввода текста?

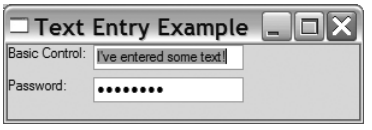

Рисунок 7.2 Примеры однострочных текстовых полей, простого и парольного

Вслед за простым выводом статического текста, мы начинаем обсуждение взаимодействия пользователя при вводе текста. Класс wxPython `wx.TextCtrl` используется в качестве виджета для текстового ввода и допускает как однострочный, так и многострочный ввод. Он может также реализовывать парольный ввод, маскируя нажимаемые клавиши. Если платформа поддерживает, то `wx.TextCtrl`

обеспечивает также расширенный текстовый вывод (rich text) с многочисленными текстовыми стилями. Рисунок 7.2 показывает пример `wx.TextCtrl` в качестве однострочного элемента управления, как с маскированием пароля, так и без него.

В следующем разделе мы покажем, как создать эти текстовые поля, а затем обсудим стилевые опции текстовых элементов управления.

Как это сделать

Листинг 7.2 содержит код, который генерирует рисунок 7.2.

Листинг 7.2 Пример однострочного `wx.TextCtrl`

```
import wx

class TextFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Text Entry Example',
                size=(300, 100))
        panel = wx.Panel(self, -1)
        basicLabel = wx.StaticText(panel, -1, "Basic Control:")
        basicText = wx.TextCtrl(panel, -1, "I've entered some text!",
                size=(175, -1))
        basicText.SetInsertionPoint(0)

        pwdLabel = wx.StaticText(panel, -1, "Password:")
        pwdText = wx.TextCtrl(panel, -1, "password", size=(175, -1),
                style=wx.TE_PASSWORD)
        sizer = wx.FlexGridSizer(cols=2, hgap=6, vgap=6)
        sizer.AddMany([basicLabel, basicText, pwdLabel, pwdText])
        panel.SetSizer(sizer)

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

Конструктор класса `wx.TextCtrl` несколько сложнее, чем у его родительского класса `wx.Window`, к нему добавлены два аргумента:

```
wx.TextCtrl(parent, id, value = "", pos=wx.DefaultPosition,
    size=wx.DefaultSize, style=0, validator=wx.DefaultValidator
    name=wx.TextCtrlNameStr)
```

Аргументы `parent`, `id`, `pos`, `size`, `style` и `name` идентичны аргументам конструктора `wx.Window`. Аргумент `value` является начальным значением отображаемого в поле текста.

Аргумент `validator` (верификатор) используется классом `wx.Validator`. Верификатор часто используется для фильтрации данных, что гарантирует ввод в

элемент управления только допустимых данных. Верификаторы подробно обсуждаются в главе 9.

Использование стилей однострочного текстового поля

В этом разделе мы начнем обсуждение некоторых специфических стилевых флагов текстовых полей. Таблица 7.3 описывает стилевые флаги, которые используются для однострочных текстовых полей.

Таблица 7.3 Стилевые флаги однострочного класса `wx.TextCtrl`

Стиль	Описание
`wx.TE_CENTER`	Текст внутри элемента управления центрируется.
`wx.TE_LEFT`	Текст внутри элемента управления выравнивается слева. Это поведение по умолчанию.
`wx.TE_NOHIDESEL`	Если Вам трудно расшифровать эту опцию, то она читается так: "не скрывать выделение". Эта опция Windows перекрывает стандартное поведение текстового виджета Windows, то есть, если виджет не имеет фокуса, выбранный текст выделяется. При выборе данной опции текст виджета будет всегда выделен. На других системах такого эффекта не будет.
`wx.TE_PASSWORD`	Вводимый текст не отображается, вместо этого он маскируется звездочками.
`wx.TE_PROCESS_ENTER`	Если указан этот флаг, то при нажатии пользователем внутри элемента управления клавиши Enter, инициируется событие ввода текста. В противном случае, нажатие этой клавиши обрабатывает или непосредственно текстовый элемент управления или диалог.
`wx.TE_PROCESS_TAB`	Если указан этот флаг, при нажатии клавиши табуляции будет создано нормальное символьное событие (обычно это означает, что табуляция будет включена в текст). Если он не указан, то клавиша табуляции будет обработана диалогом как обычная навигация между элементами управления.
`wx.TE_READONLY`	Текст элемента управления будет доступен только для чтения и не может быть модифицирован пользовательским вводом.
`wx.TE_RIGHT`	Текст внутри элемента управления выравнивается справа.

Подобно другим стилевым флагам, данные флаги могут объединяться при помощи оператора |, хотя три флага выравнивания взаимно исключают друг друга.

Текстовый элемент управления автоматически обслуживает нажатия пользователем клавиш и события мыши, которые приводят к добавлению текста и перемещению позиции ввода. Разрешены следующие общепринятые управляющие комбинации:

- <ctrl-x> вырезать
- <ctrl-c> копировать
- <ctrl-v> вставить
- <ctrl-z> отменить

7.1.3 Как изменить текст без ввода со стороны пользователя?

Помимо изменения вида текста, основанного на вводе пользователя, `wx.TextCtrl` реализует множество методов, которые изменяют текст на экране непосредственно из Вашей программы. Вы можете изменить текст полностью или только переместить позицию ввода в другое место текста. Таблица 7.4 перечисляет методы класса `wx.TextCtrl`, предназначенные для обработки текста.

Таблица 7.4 Методы обработки текста `wx.TextCtrl`

Метод	Описание
`AppendText(text)`	Добавляет текстовый аргумент в конец текстового элемента управления. Позиция ввода перемещается на конец элемента управления.
`Clear()`	Сбрасывает текстовое значение элемента управления в "". Вызывает генерацию события обновления текста.
`EmulateKeyPress(event)`	Данное событие нажатия клавиши вставляет в элемент управления связанный с этим событием символ, имитируя тем самым фактическое нажатие.

`GetInsertionPoint()`
`SetInsertionPoint(pos)`
`SetInsertionPointEnd()`
	Получение или установка целочисленного индекса текущей позиции ввода, или другими словами, индекса, где размещается следующий вставляемый символ. Начало элемента управления имеет индекс 0.

`GetRange(from, to)`	Возвращает строку между указанными целыми позициями элемента управления.

`GetSelection()`
`GetStringSelection()`
`SetSelection(from, to)`
	`GetSelection()` возвращает кортеж (start, end) с индексами выделенного в данный момент текста. `GetStringSelection()` возвращает выделенную строку текста. Метод `SetSelection(from, to)` устанавливает выделение по его конечным точкам.

`GetValue()` `SetValue(value)`	Метод `SetValue()` изменяет значение элемента управления целиком. `GetValue()` возвращает целиком всю его строку.
`Remove(from, to)`	Удаляет указанный диапазон текста.
`Replace(from, to, value)`	Замещает указанный диапазон новым значением. Это может изменить длину текста.
`WriteText(text)`	Аналогичен методу `AppendText()`, за исключением того, что новый текст помещается в текущую позицию ввода.

Данные методы особенно полезны, когда Вы работаете с элементом управления в режиме «только для чтения», или если Вам необходимо изменить текст элемента управления с помощью событий, отличных от происходящих при нажатии пользователем клавиш.

7.1.4 Как создать многострочный или стилизованный текстовый элемент управления?

При помощи стилевого флага wx.TE_MULTILINE Вы можете создать многострочный текстовый элемент управления. Если виджет поддерживает стили, Вы можете изменить стили шрифта и цвета в тексте этого элемента управления (иногда его называют расширенным текстом - rich text). На других платформах вызовы, которые устанавливают стили, просто игнорируются. Рисунок 7.3 отображает пример многострочных текстовых элементов управления.

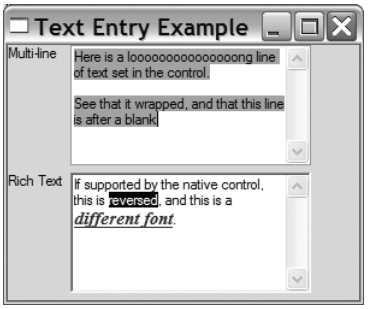

Рисунок 7.3 Примеры многострочных текстовых полей, простого и с расширенным текстом

Листинг 7.3 содержит код, использованный для создания рисунка 7.3. Обычно, создание многострочного элемента управления управляется установкой стилевого флага wx.TE_MULTILINE. Далее в этом разделе мы обсудим использование стилей расширенного текста.

Листинг 7.3 Создание многострочного текстового элемента управления

```
import wx

class TextFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Text Entry Example',
                size=(300, 250))
        panel = wx.Panel(self, -1)
        multiLabel = wx.StaticText(panel, -1, "Multi-line")
        multiText = wx.TextCtrl(panel, -1,
               "Here is a looooooooooooooong line "
               "of text set in the control.\n\n"
               "See that it wrapped, and that "
               "this line is after a blank",
        size=(200, 100), style=wx.TE_MULTILINE)
        multiText.SetInsertionPoint(0)

        richLabel = wx.StaticText(panel, -1, "Rich Text")
        richText = wx.TextCtrl(panel, -1,
               "If supported by the native control, "
               "this is reversed, and this is a different font.",
               size=(200, 100),
```

Создание текстового поля

Установка позиции курсора

Создание расширенного текста

```
            style=wx.TE_MULTILINE|wx.TE_RICH2)
        richText.SetInsertionPoint(0)
        richText.SetStyle(44, 52, wx.TextAttr("white", "black"))
        points = richText.GetFont().GetPointSize()
        f = wx.Font(points + 3, wx.ROMAN,
            wx.ITALIC, wx.BOLD, True)
        richText.SetStyle(68, 82, wx.TextAttr("blue",
            wx.NullColour, f))
        sizer = wx.FlexGridSizer(cols=2, hgap=6, vgap=6)
        sizer.AddMany([multiLabel, multiText, richLabel, richText])
        panel.SetSizer(sizer)

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

Установка текстовых стилей

Создание шрифта

Установка стиля нового шрифта

Использование стилей расширенного текста

Кроме wx.TE_MULTILINE имеются и другие стилевые флаги, которые имеют значение в контексте многострочного или расширенного текстового элемента управления. Таблица 7.5 перечисляет эти стили окна.

Таблица 7.5 Стилевые биты многострочного `wx.TextCtrl`

Стиль	Описание
`wx.HSCROLL`	Если текстовый элемент управления многострочный и указан этот флаг, длинные строки вместо переноса будут горизонтально прокручиваться. В GTK+ эта опция игнорируется.
`wx.TE_AUTO_URL`	Если установлен флаг расширенного текста и платформа это поддерживает, данный стиль разрешает генерацию событий при движении мыши и щелчках в тексте на URL.
`wx.TE_DONTWRAP`	Другое имя для `wx.HSCROLL`.
`wx.TE_LINEWRAP`	Противопоставлен флагу `wx.TE_WORDWRAP`. Длинные строки могут быть разорваны для переноса на любом символе. Некоторые операционные системы игнорируют этот флаг.
`wx.TE_MULTILINE`	Текстовый элемент управления будет отображать множество строк.
`wx.TE_RICH`	В качестве основного виджета под Windows будет использоваться элемент управления с расширенным текстом. Данная возможность позволяет использовать стилизованный текст.
`wx.TE_RICH2`	В качестве основного виджета под Windows будет использоваться последняя версия элемента управления расширенного текста.

wx.TE_WORDWRAP	Противопоставлен флагу wx.TE_LINEWRAP. Требующие переноса строки будут разорваны на границе слова. На многих системах эта опция игнорируется.

Помните, что стилевые флажки могут быть объединены, например, многострочное поле расширенного текста в последнем примере объявлено со стилем wx.TE_MULTILINE | wx.TE_RICH2. Использующиеся в виджете wx.TextCtrl текстовые стили являются экземплярами класса wx.TextAttr. Отдельный экземпляр wx.TextAttr включает цвет текста, цвет фона и шрифт, определяемые в конструкторе следующим образом:

```
wx.TextAttr(colText, colBack=wx.NullColor, font=wx.NullFont)
```

Цвета текста и фона являются объектами wxPython типа wx.Color и могут определяться при помощи строки идентификатора цвета или кортежа со значениями цветовых составляющих (красный, зеленый, синий). Значение wx.NullColor указывает, что будет использован существующий цвет фона элемента управления. Шрифт представлен объектом wx.Font и будет рассмотрен в следующем разделе. Объект wx.NullFont указывает на то, что должен быть использован установленный по умолчанию текущий шрифт.

Класс wx.TextAttr имеет методы для получения атрибутов текста GetBackgroundColour(), GetFont(), GetTextColour(), а также логические методы существования HasBackgroundColour(), HasFont() и HasTextColour(). Если атрибут содержит значение по умолчанию, методы существования возвращают False. Метод IsDefault() возвращает True, если значение по умолчанию содержат все три атрибута. Поскольку экземпляры wx.TextAttr являются неизменяемыми, класс wx.TextAttr не имеет методов установки. Чтобы изменить стиль текста, Вы должны создать его экземпляр.

Для того, чтобы использовать текстовый стиль, вызовите метод SetDefaultStyle(style) или SetStyle(start, end, style). Первый метод устанавливает текущий стиль элемента управления. После этого любой вставляемый в элемент управления текст, как путём набора, так и при помощи методов AppendText() или WriteText(), будет отображен в данном стиле. Если какой-либо из атрибутов стиля имеет значение по умолчанию, текущая величина этого стиля сохраняется. Однако если значение по умолчанию имеют *все* атрибуты стиля, восстанавливается стиль по умолчанию. Метод SetStyle() аналогичен, но воздействует на текст непосредственно между позициями start и end. Атрибуты по умолчанию в аргументе стиля определяются путем проверки в элементе управления текущего стиля по умолчанию. Листинг 7.3 использует следующую строчку кода для назначения определенных цветов нескольким символам текста:

```
richText.SetStyle(44, 52, wx.TextAttr("white", "black"))
```

Цвет фона становится белым, а цвет текста для этих символов становится черным.

Таблица 7.6 перечисляет методы `wx.TextCtrl`, которые полезны при работе с многострочными элементами управления и расширенным текстом.

Таблица 7.6 Методы `wx.TextCtrl` для многострочного и стилизованного текста

Метод	Описание
`GetDefaultStyle()` `SetDefaultStyle(style)`	Смотрите начало этого раздела с описанием встроенных стилей.
`GetLineLength(lineNo)`	Возвращает целую длину указанной строки.
`GetLineText(lineNo)`	Возвращает текст указанной строки.
`GetNumberOfLines()`	Возвращает число строк в элементе управления. Для однострочных возвращает 1.
`IsMultiLine()` `IsSingleLine()`	Булевые методы для определения состояния элемента управления.
`PositionToXY(pos)`	Для заданной целой позиции внутри текста возвращает кортеж (столбец, строка) индексов позиции. Столбец и строка индексируются, начиная с 0.
`SetStyle(start, end, style)`	Изменяет стиль указанного диапазона текста.
`ShowPosition(pos)`	Заставляет многострочный элемент управления прокручиваться, так что данная позиция будет оставаться в поле зрения.
`XYToPosition(x, y)`	Противоположный методу `PositionToXY` — для указанных столбца и строки возвращает целую позицию.

Работа со стилями становится более гибкой, если в своей системе Вы можете использовать произвольные шрифты. Далее мы покажем Вам, как создавать и использовать экземпляры шрифтов.

7.1.5 *Как создать шрифт?*

Шрифты определены как экземпляры класса `wx.Font`. Вы имеете доступ к любому шрифту, который установлен и доступен в Вашей системе. Для того чтобы создать экземпляр шрифта, используйте следующий конструктор:

```
wx.Font(pointSize, family, style, weight, underline=False,
    faceName="", encoding=wx.FONTENCODING_DEFAULT)
```

Параметр `pointSize` определяет целый размер шрифта в точках. Параметр `family` используется, чтобы быстро определить шрифт без необходимости указания его фактического имени. Конкретный шрифт выбирается в зависимости от доступных системных и специальных шрифтов. Примеры возможных семейств шрифтов показаны в таблице 7.7. Конкретные шрифты, которые Вы получите, будут зависеть от Вашей системы.

Таблица 7.7 Образцы существующих семейств шрифтов

Шрифт	Описание
wx.DECORATIVE	Формальный старо-английский стиль шрифта
wx.DEFAULT	Системный шрифт по умолчанию
wx.MODERN	Моноширинный шрифт (с фиксированным шагом)
wx.ROMAN	Шрифт с засечками, обычно приблизительно похож на Times New Roman
wx.SCRIPT	Прописной или рукописный шрифт
wx.SWISS	Шрифт sans-serif , обычно приблизительно соответствует Helvetica или Arial.

Параметр `style` указывает характер наклона шрифта: `wx.NORMAL`, `wx.SLANT` или `wx.ITALIC`. Аналогично, параметр `weight` (вес) указывает толщину шрифта: `wx.NORMAL`, `wx.LIGHT` или `wx.BOLD`. Константы ведут себя здесь, как это следует из их имени. Параметр `underline` (подчеркивание) работает только в системах Windows, и если он установлен в `True`, шрифт необходимо подчеркивать. Используйте аргумент `faceName`, чтобы указать системное имя отображаемого Вами шрифта.

Параметр `encoding` позволяет Вам выбрать тот или иной вариант кодирования, который соотносит внутренние символы и шрифт дисплейных символов. Данное кодирование *не* является кодированием Unicode, это просто другое применяемое в wxPython 8-битовое кодирование. В большинстве случаев Вам достаточно использовать кодирование по умолчанию.

Для того чтобы получить список доступных в системе шрифтов и сделать их доступными пользователю, используйте следующим образом специальный класс `wx.FontEnumerator`:

```
e = wx.FontEnumerator()
e.EnumerateFacenames()
fontList = e.GetFacenames()
```

Для ограничения списка по ширине, измените первую строку на:

```
e = wx.FontEnumerator(fixedWidth=True).
```

7.1.6 *Можно ли стилизировать текст, если платформа не поддерживает расширенный текст (rich text)?*

Да. В wxPython имеется кросс-платформенный виджет стилизованного текста, называющийся `wx.stc.StyledTextCtrl`, который является надстройкой к компоненту расширенного текста Scintilla. Поскольку Scintilla не является частью wxWidgets и представляет собой включенный в wxPython отдельный независимый компонент, он не имеет такого же API, как и у рассмотренных нами классов.

Детальное обсуждение `wx.stc.StyledCtrl` выходит за рамки нашей книги, тем не менее, Вы можете найти соответствующую документацию по адресу:

http://wiki.wxpython.org/index.cgi/wxStyledTextCtrl.

7.1.7 *Что если мой текстовый элемент управления не подходит для моей строки?*

Применяя многострочный виджет `wx.TextCtrl`, Вы должны иметь элементарное представление относительно способа, которым текстовый элемент управления хранит строку текста. Многострочный текст в `wx.TextCtrl` хранится, используя в качестве разделителя строк символ `\n`. Даже если некоторые системы в качестве разделителя строк используют другие символьные комбинации, такой способ реально независим от типа используемой операционной системы. Когда Вы извлекаете строку методом `GetValue()`, системный разделитель строк восстанавливается и Вам не нужно беспокоиться об обратном ручном преобразовании. Преимущество данного способа состоит в том, что текст внутри элемента управления не зависит от каких-либо особенностей операционной системы.

Недостаток состоит в том, что длина строки и индексы внутри и за пределами текстового элемента управления могут отличаться. Например, если вы имеете дело с системой Windows, где в качестве разделителя строки применяется `\r\n`, длина строки, возвращенной методом `GetValue()`, будет длиннее, чем сообщаемый методом `GetLastPosition()` конец строки в элементе управлении. Добавляя следующие две строки в листинг 7.3,

```
print "getValue", len(multiText.GetValue())
print "lastPos", multiText.GetLastPosition()
```

мы получим следующий результат в системе Unix:

```
getValue 119
lastPos 119
```

и следующий результат в системе Windows:

```
getValue 121
lastPos 119
```

Смысл в том, что для обратной ссылки на оригинальную строку Вы не должны использовать позиционные индексы многострочного текстового элемента управления, они должны использоваться только как аргументы в других методах wx.TextCtrl. Для подстроки текста внутри элемента управления используйте методы `GetRange()` или `GetSelectedText()`. К тому же, не пересекайте индексы при перестановке (реверсировании); не используйте индексы оригинальной строки для обратной ссылки в текстовый элемент управления.

Ниже приведен пример неправильного способа получения 10 символов, следующих за позицией ввода:

```
aLongString = """Any old
    multi line string
    will do here.
    Just as long as
    it is multiline"""
text = wx.TextCtrl(panel, -1, aLongString, style=wx.TE_MULTILINE)
x = text.GetInsertionPoint()
selection = aLongString[x : x + 10] ### ЭТО НЕПРАВИЛЬНО
```

Последняя строка должна быть снабжена комментарием для систем Windows или Mac, поскольку она использует x (позиция точки ввода текстового поля) в качестве индекса оригинальной строки. Для того чтобы вернуть правильные символы в системах Windows или Mac, последняя строка должна быть записана следующим образом:

```
selection = text.GetRange(x, x + 10)
```

7.1.8 Как реагировать на события обработки текста?

Вам может реально пригодиться масса генерируемых виджетами wx.TextCtrl командных событий. Все эти события соединяются с конкретным текстовым виджетом, а для того чтобы захватывать событие, его необходимо передать в метод Bind:

```
frame.Bind(wx.EVT_TEXT, frame.OnText, text)
```

Таблица 7.8 описывает командные события.

Таблица 7.8 События виджетов wx.TextCtrl

Событие	Описание
EVT_TEXT	Генерируется при изменении текста элемента управления. Это событие генерируется как в ответ на ввод пользователя, так и при программном изменении текста методом SetValue().
EVT_TEXT_ENTER	Генерируется при установленном стиле wx.TE_PROCESS_ENTER, в случае нажатия пользователем в элементе управления клавиши Enter.
EVT_TEXT_URL	Это событие вырабатывается на системах Windows при установленных стилях wx.TE_RICH или wx.TE_RICH2, а также установленном стиле wx.TE_AUTO_URL в случае, когда происходит событие мыши над URL в тексте элемента управления.
EVT_TEXT_MAXLEN	Это событие вырабатывается, когда пользователь пытается ввести строку, которая длиннее, чем заданная методом SetMaxLength()максимальная длина строки элемента управления. Вы можете использовать это событие, например, для отображения пользователю предупреждающего сообщения.

Далее мы обсудим элементы управления, основное назначение которых состоит в получении ввода от мыши. Простейший из них - кнопка.

7.2 Работа с кнопками

В wxPython существует большое количество различных типов кнопок. В этом разделе мы рассмотрим текстовые кнопки, кнопки с битовыми изображениями, кнопки-переключатели и типовые (generic) кнопки.

7.2.1 Как создать кнопку?

В части 1 мы уже рассматривали несколько примеров кнопок, поэтому здесь мы только кратко изложим их основы. Рисунок 7.4 показывает простую кнопку.

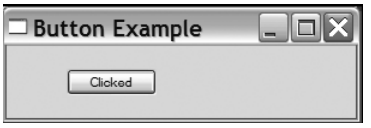

Рисунок 7.4 Пример простой кнопки

Принцип использования кнопки очень прост. Листинг 7.4 содержит код этого простого примера кнопки.

Листинг 7.4 Создание и отображение простой кнопки

```
import wx

class ButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Button Example',
                size=(300, 100))
        panel = wx.Panel(self, -1)
        self.button = wx.Button(panel, -1, "Clicked", pos=(50, 20))
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
        self.button.SetDefault()

    def OnClick(self, event):
        self.button.SetLabel("Clicked")

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

Конструктор `wx.Button` подобен конструкторам, которые мы уже видели ранее:

```
wx.Button(parent, id, label, pos, size=wxDefaultSize, style=0,
        validator, name="button")
```

Характерный для `wx.Button` аргумент `label` представляет собой отображаемый на кнопке текст. Он может быть изменен при выполнении программы методом `SetLabel()`, а получен методом `GetLabel()`. Два других полезных метода -

GetDefaultSize(), который возвращает предложенный системой встроенный размер кнопки (полезен для согласования фреймов по ширине), и SetDefault(), который указывает, что данная кнопка диалога или фрейма будет кнопкой по умолчанию. Кнопка по умолчанию часто выглядит иначе чем другие кнопки и когда диалог имеет фокус, обычно активируется нажатием клавиши Enter.

Класс wx.Button имеет один кросс-платформенный флаг стиля wx.BU_EXACTFIT. Если он указан, кнопка не использует в качестве минимального размера системный размер по умолчанию, а вместо этого уменьшает свой размер к минимально допустимому размеру для вписывания надписи. Если виджетом поддерживается, то Вы можете изменять выравнивание надписи внутри кнопки при помощи флагов wx.BU_LEFT, wx.BU_RIGHT, wx.BU_TOP и wx.BU_BOTTOM. Каждый флаг выравнивает надпись точно по той стороне, которая как Вы понимаете, указана в имени флага. В части 1 мы увидели, что wx.Button инициирует при нажатии одно командное событие типа EVT_BUTTON.

7.2.2 Как создать кнопку с битовым изображением?

Иногда необходимо изобразить на кнопке вместо надписи некоторую картинку, например, как на рисунке 7.5.

Чтобы создавать кнопку с изображением в wxPython используется класс wx.BitmapButton. Как показано в листинге 7.5, управляющий виджетом wx.BitmapButton код очень похож на код обычной кнопки.

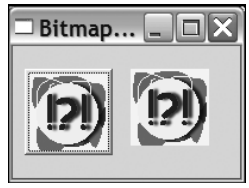

Рисунок 7.5 Демонстрация кнопок с изображением. Левая кнопка имеет эффект 3D.

Листинг 7.5 Создание кнопок с изображением

```
import wx

class BitmapButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Bitmap Button Example',
                size=(200, 150))
        panel = wx.Panel(self, -1)
        bmp = wx.Image("bitmap.bmp",
            wx.BITMAP_TYPE_BMP).ConvertToBitmap()
        self.button = wx.BitmapButton(panel, -1, bmp, pos=(10, 20))
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
        self.button.SetDefault()
        self.button2 = wx.BitmapButton(panel, -1, bmp, pos=(100, 20),
            style=0)
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button2)

    def OnClick(self, event):
        self.Destroy()

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

```
frame = BitmapButtonFrame()
frame.Show()
app.MainLoop()
```

Основное отличие в том, что для кнопки с битовым изображением Вам нужно назначить не надпись, а соответствующее битовое изображение. Во всем остальном конструктор и большая часть кода идентичны варианту с текстовой кнопкой. Кнопка с битовым изображением вырабатывает при нажатии всё тоже событие `EVT_BUTTON`.

Кнопки с битовыми изображениями имеют две интересных возможности. Во-первых, это установленный по умолчанию стилевой флаг `wx.BU_AUTODRAW`. Если этот флаг установлен, битовое изображение кнопки будет окружено 3D-границей, что сделает её похожей на текстовую кнопку (левая кнопку на рисунке 7.5), и кнопка будет на несколько пикселей больше, чем оригинальное битовое изображение. Если этот флаг выключен, битовое изображение будет нарисовано просто как кнопка без границы. У правой кнопки на рисунке 7.5 `style=0`, что снимает установку по умолчанию, и у неё нет объемного эффекта 3D.

Просто передайте wxPython битовое изображение для основного экрана, и wxPython автоматически создаст стандартные производные битовых изображений для нажатой кнопки, кнопки имеющей фокус и блокированной кнопки. Если стандартное поведение Вас не устраивает, Вы можете явно указать wxPython необходимые битовые изображения следующими методами: `SetBitmapDisabled()`, `SetBitmapFocus()`, `SetBitmapLabel()` и `SetBitmapSelected()`. Каждый из этих методов получает объект `wx.Bitmap`, и вместе с тем имеются соответсвующие функции чтения назначенных битовых изображений.

Используя стандартную C++ библиотеку wxWidgets, Вы не можете комбинировать битовое изображение и текст. У Вас имеется единственная возможность создать битовое изображение, содержащее текст. Тем не менее, как мы увидим при обсуждении типовой кнопки, в wxPython имеется дополнительная возможность, которая разрешает эту проблему.

7.2.3 Как создать кнопку-переключатель?

Вы можете создать *кнопку-переключатель* при помощи класса `wx.ToggleButton`. Кнопка-переключатель выглядит точно также, как и текстовая кнопка, но ведет себя подобно флажку (checkbox), поскольку она даёт визуальное восприятие выбранного или невыбранного состояния. Другими словами, когда Вы нажимаете кнопку-переключатель, она показывает свое состояние, оставаясь на вид нажатой, пока Вы снова её не щелкнете.

Имеется всего два различия между `wx.ToggleButton` и его родительским классом wx.Button:

- `wx.ToggleButton` при нажатии посылает событие `EVT_TOGGLEBUTTON`.

- `wx.ToggleButton` имеет методы `GetValue()` и `SetValue()`, которые управляют двоичным состоянием кнопки.

Кнопки-переключатели могут представлять полезную и привлекательную альтернативу переключателям-флажкам (checkbox), особенно в инструментальных панелях. Помните, что используя готовые объекты wxWidgets, у Вас нет возможноти объединить кнопку-переключатель и кнопку с битовым изображением, но wxPython имеет обеспечивающий это поведение класс типовой кнопки, который мы опишем в следующем разделе.

7.2.4 *Что такое типовая (generic) кнопка и почему её необходимо использовать?*

Типовая кнопка является кнопочным виджетом, который полностью воспроизведен на Python, игнорируя использование стандартного системного виджета. Общим для кнопок с битовым изображением и кнопок-переключателей выступает родительский класс `GenButton` из модуля `wx.lib.buttons`.

Для использования типовых кнопок имеется несколько причин:

- Типовая кнопка в отличие от стандартных кнопок выглядит на разных платформах практически одинаково. Хотя с другой стороны, в одной и той же системе вид типовых кнопок может несколько отличаться от вида стандартных кнопок.
- Применяя типовые кнопки, Вы имеете больше возможностей по управлению её видом, и можете изменять её атрибуты теми способами, которые могут быть недоступными для стандартных элементов управления, например, изменять ширину 3D края и цвет.
- Семейство типовых кнопок позволяет использовать возможности, которых нет у кнопок wxWidgets. Например, имеется кнопка `GenBitmapTextButton`, которая допускает комбинацию текстовой надписи и битового изображения, а кнопка `GenBitmapToggleButton` реализует переключатель с битовым изображением.
- Если Вы создаете свой класс кнопки, то в качестве базового класса лучше использовать типовую кнопку. Так как её код и параметры написаны на Python, они лучше всего подходят для исследования и воспроизведения при создании нового подкласса.

Рисунок 7.6 показывает типовую кнопку в работе.

Листинг 7.6 содержит код для рисунка 7.6. Второй оператор импорта – `import wx.lib.buttons as buttons` – делает доступными классы общих кнопок.

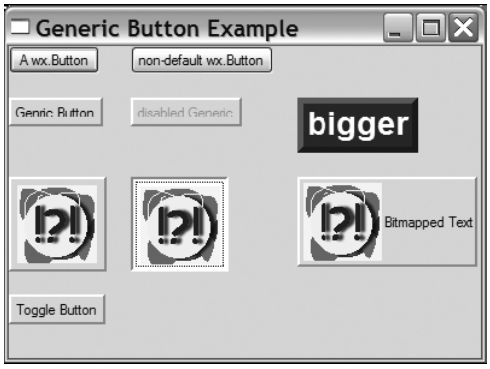

Рисунок 7.6
Типовые кнопки. Верхняя строка содержит для сравнения обычные кнопки. Показаны различные сочетания цветов, кнопка с битовым изображением, кнопка-переключатель с битовым изображением и кнопка с текстом и битовым изображением

Листинг 7.6 Создание и использование типовых кнопок wxPython

```
import wx
import wx.lib.buttons as buttons

class GenericButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Generic Button Example',
                size=(500, 350))
        panel = wx.Panel(self, -1)

        sizer = wx.FlexGridSizer(1, 3, 20, 20)
        b = wx.Button(panel, -1, "A wx.Button")
        b.SetDefault()
        sizer.Add(b)

        b = wx.Button(panel, -1, "non-default wx.Button")
        sizer.Add(b)
        sizer.Add((10,10))

        b = buttons.GenButton(panel, -1, 'Generic Button')
        sizer.Add(b)

        b = buttons.GenButton(panel, -1, 'disabled Generic')
        b.Enable(False)
        sizer.Add(b)

        b = buttons.GenButton(panel, -1, 'bigger')
        b.SetFont(wx.Font(20, wx.SWISS, wx.NORMAL, wx.BOLD, False))
        b.SetBezelWidth(5)
        b.SetBackgroundColour("Navy")
        b.SetForegroundColour("white")
        b.SetToolTipString("This is a BIG button...")
        sizer.Add(b)

        bmp = wx.Image("bitmap.bmp",
            wx.BITMAP_TYPE_BMP).ConvertToBitmap()
        b = buttons.GenBitmapButton(panel, -1, bmp)
        sizer.Add(b)
```

Базовая типовая кнопка

Блокированная типовая кнопка

Кнопка с заданным пользователем размером и цветом

Типовая кнопка с битовым изображением

```
        b = buttons.GenBitmapToggleButton(panel, -1, bmp)
        sizer.Add(b)

        b = buttons.GenBitmapTextButton(panel, -1, bmp,
            "Bitmapped Text", size=(175, 75))
        b.SetUseFocusIndicator(False)
        sizer.Add(b)

        b = buttons.GenToggleButton(panel, -1, "Toggle Button")
        sizer.Add(b)

        panel.SetSizer(sizer)

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

Типовая кнопка-переключатель с битовым изображением

Кнопка с битовым изображением и текстом

Типовая кнопка-переключатель

Использование типовой кнопки в листинге 7.6 очень похоже на стандартную кнопку. Типовые кнопки как и стандартные кнопки генерируют те же события `EVT_BUTTON` и `EVT_TOGGLEBUTTON`. Для изменения толщины 3D кромки типовая кнопка имеет методы `GetBevelWidth()` и `SetBevelWidth()`. Они использованы для большой кнопки рисунка 7.6.

Класс типовой кнопки с битовым изображением `GenBitmapButton` работает подобно обычной версии wxPython. `GenBitmapTextButton` принимает в конструкторе сначала битовое изображение, а затем текст. Типовые кнопки предоставляют три класса переключателей: `GenToggleButton`, `GenBitmapToggleButton` и `GenBitmapTextToggleButton`. Все эти три класса аналогичны версиям без переключателей, а управление состоянием переключателя производится при помощи методов `GetToggle()` и `SetToggle()`.

В следующем разделе мы обсудим возможности, которые позволят Вашему пользователю вводить или просматривать числовые величины.

7.3 Ввод и отображение числовых значений

Иногда Вам необходимо графически изобразить числовую информацию, или Вам нужно, чтобы пользователь ввёл числовое значение без использования клавиатуры. В этом разделе мы исследуем следующие инструментальные средства wxPython для ввода и отображения числовых данных: ползунок (slider), счетчик (spinner box) и индикатор прогресса (gauge).

7.3.1 Как создать ползунок (slider)?

Ползунок представляет собой виджет, который позволяет пользователю выбирать число в дипазоне значений, перемещая маркер элемента управления по горизонтали или вертикали. В wxPython ползунок реализует класс `wx.Slider`,

содержащий доступный только для чтения текст текущей величины ползунка. Рисунок 7.7 приводит примеры вертикального и горизонтального ползунков. Основное использование ползунка выглядит довольно просто, но вместе с этим имеется множество событий, которые Вы можете к нему присоединить.

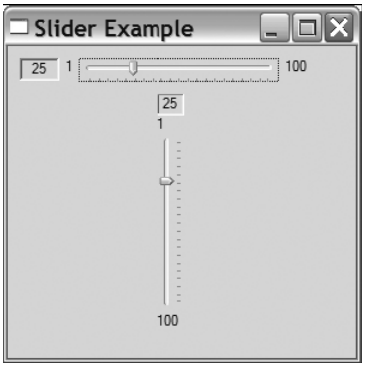

Рисунок 7.7 Вертикальный и горизонтальный виджеты `wx.Slider`, использующие стилевой флаг `wx.SL_LABELS`

Как использовать ползунок?

Как показано в листинге 7.7, ползунок может управлять единственной величиной.

Листинг 7.7 Код вывода горизонтального и вертикального ползунков

```
import wx

class SliderFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Slider Example',
              size=(340, 320))
        panel = wx.Panel(self, -1)
        self.count = 0
        slider = wx.Slider(panel, 100, 25, 1, 100, pos=(10, 10),
              size=(250, -1),
              style=wx.SL_HORIZONTAL | wx.SL_AUTOTICKS | wx.SL_LABELS)
        slider.SetTickFreq(5, 1)
        slider = wx.Slider(panel, 100, 25, 1, 100, pos=(125, 50),
              size=(-1, 250),
              style=wx.SL_VERTICAL | wx.SL_AUTOTICKS | wx.SL_LABELS)
        slider.SetTickFreq(20, 1)

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

Обычно, при использовании класса `wx.Slider`, все, что Вам требуется - это конструктор, который отличается от аналогичных вызовов следующим образом:

```
wx.Slider(parent, id, value, minValue, maxValue,
        pos=wxDefaultPosition, size=wx.DefaultSize,
        style=wx.SL_HORIZONTAL, validator=wx.DefaultValidator,
        name="slider")
```

Параметр `value` определяет начальное значение ползунка, а параметры `minValue` и `maxValue` – его предельные значения.

Работа со стилями ползунка

Как показано в таблице 7.9, стили ползунка управляют его размещением и ориентацией.

Таблица 7.9 Стили `wx.Slider`

Стиль	Описание
`wx.SL_AUTOTICKS`	Если установлен, ползунок будет отображать отметки. Частота повторения отметок задается методом `SetTickFreq`.
`wx.SL_HORIZONTAL`	Ползунок будет горизонтальным. Этот стиль установлен по умолчанию.
`wx.SL_LABELS`	Если установлен, ползунок будет отображать подписи для минимальной и максимальной величины, а текущее значение будет доступно только для чтения. Текущее значение на некоторых платформах может не отображаться.
`wx.SL_LEFT`	Для вертикального ползунка отметки будут установлены с его левой стороны.
`wx.SL_RIGHT`	Для вертикального ползунка отметки будут установлены с его правой стороны.
`wx.SL_TOP`	Для горизонтального ползунка отметки будут установлены сверху.
`wx.SL_VERTICAL`	Ползунок будет вертикальным.

Если Вы хотите, чтобы изменения значения ползунка влияло на другую часть Вашего приложения, Вы можете использовать для этого несколько имеющихся событий. Эти события идентичны тем, которые генерируются оконной полосой прокрутки и подробно описываются в разделе о прокрутке главы 8.

Таблица 7.10 перечисляет относящиеся к ползунку установочные свойства. Каждый установщик имеет соответствующий `Get`-метод, описания же в данной таблице ссылаются только на установочные методы.

Таблица 7.10 Установочные методы ползунка

Метод	Описание
`SetRange(minValue, maxValue)` `GetRange()`	Устанавливает диапазон крайних значений ползунка.
`SetTickFreq(n, pos)` `GetTickFreq()`	Устанавливает частоту повторения отметок при помощи аргумента n. Аргумент pos фактически не используется, но он необходим. Установите его в 1.

`SetLineSize(lineSize)` `GetLineSize()`	Устанавливает величину, на которую изменяется показание ползунка, если Вы смещаете его движок на одну линию при помощи клавиши стрелки.
`SetPageSize(pageSize)` `GetPageSize()`	Устанавливает величину, на которую изменяется показание ползунка, если Вы смещаете его движок на одну страницу при помощи клавиш Page Up или Page Down.
`SetValue(value)` `GetValue()`	Устанавливает значение ползунка.

Хотя ползунки обеспечивают быстрое визуальное представление, где значение располагается вдоль возможного дипазона, у них все же есть несколько недостатков. Они занимают много места в своем основном направлении, и, используя мышь, трудно устанавливать движок точно, особенно если дипазон довольно большой, или если пользователь имеет ограниченные возможности. Обе этих проблемы решает обсуждаемый в следующем разделе виджет – счетчик (spinner).

7.3.2 Как создать счетчик (spinner)?

Счетчик сочетает в себе текстовое поле и пару кнопок со стрелками, которые устанавливают числовое значение, и если Вы ограничены в экранном пространстве, он представляет хорошую альтернативу ползунку. Рисунок 7.8 показывает пример счетчика wxPython.

Рисунок 7.8 Пример счетчика wxPython

В wxPython кнопками счетчика и ассоциированным текстовым полем управляет класс `wx.SpinCtrl`. В следующем разделе мы создадим счетчик.

Создание счетчика

Используйте виджет `wx.SpinCtrl` для изменения значения нажатием кнопок или непосредственным вводом в текстовое поле. Набранный в элементе управления нечисловой текст игнорируется, однако пока не будет нажата кнопка, значение элемента управления не станет прежним. Числовое значение вне диапазона счетчика трактуется в качестве максимальной или минимальной величины, но пока Вы не нажмете кнопку счетчика, крайнее значение диапазона не устанавливается. Использование виджета `wx.SpinCtrl` показывает листинг 7.8.

Листинг 7.8 Использование `wx.SpinCtrl`

```
import wx

class SpinnerFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Spinner Example',
```

```
                size=(150, 100))
        panel = wx.Panel(self, -1)
        sc = wx.SpinCtrl(panel, -1, "", (30, 20), (80, -1))
        sc.SetRange(1,100)
        sc.SetValue(5)

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

Почти вся сложность в управлении счетчиком сосредоточена в конструкторе, который имеет несколько аргументов:

```
wx.SpinCtrl(parent, id=-1, value=wx.EmptyString,
        pos=wx.DefaultPosition, size=wx.DefaultSize,
        style=wx.SP_ARROW_KEYS, min=0, max=100, initial=0,
        name="wxSpinCtrl")
```

Первая часть этого конструктора аналогична другим конструкторам `wx.Window`. Однако, аргумент `value` в данном случае фиктивный. Установите начальное значение счетчика при помощи аргумента `initial`, а для задания его диапазона используйте аргументы `min` и `max`.

Виджет `wx.SpinCtrl` имеет два стилевых флага. По умолчанию объявлен флаг `wx.SP_ARROW_KEYS`, поскольку он позволяет пользователю изменять значение счетчика с клавиатуры при помощи клавиш Up и Down. Стиль `wx.SP_WRAP` заставляет зациклить значение счетчика, и если Вы достигните границы в одном предельном значении, произойдет автоматический переход на другое предельное значение. Вы также можете реагировать на событие `EVT_SPINCTRL`, которое генерируется всякий раз, когда значение счетчика изменяется (даже, если изменение происходит посредством текстового ввода). Если текст изменяется, то подобно тому, как будто Вы используете отдельный текстовый элемент управления, будет инициировано событие `EVT_TEXT`.

Как показано в листинге 7.8, Вы можете установить диапазон и значение счетчика при помощи методов `SetRange(minVal, maxVal)` и `SetValue(value)`. Функция `SetValue()` может принимать строку или целое значение. Для того чтобы прочитать значения счетчика используйте методы `GetValue()` (который возвращает целое), `GetMin()` и `GetMax()`.

Когда Вам необходим больший контроль над поведением счетчика, например, использовать значения с плавающей запятой или список строк, Вы можете комбинировать `wx.SpinButton` вместе с `wx.TextCtrl` и устроить их взаимодействие. Разместите два этих элемента управления рядом и обрабатывайте события `EVT_SPIN` от `wx.SpinButton`, обновляя при этом значение в `wx.TextCtrl`.

7.3.3 Как создать индикатор прогресса (progress bar)?

Если Вы хотите наглядно отобразить числовую величину, не допуская её изменения со стороны пользователя, используйте важный виджет wxPython

wx.Gauge. Примером такой числовой величины является показанный на рисунке 7.9 *индикатор прогресса*.

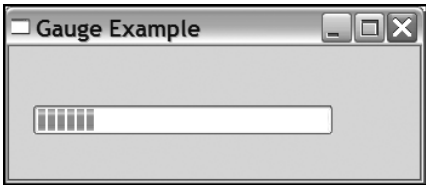

Рисунок 7.9 Виджет wx.Gauge отображает некоторый процесс

Листинг 7.9 содержит код, который создает этот рисунок. В отличие от многочисленных примеров этой главы, здесь мы добавили обработчик события. Следующий код регулирует величину индикатора в течение времени простоя системы, вызывая его циклическое заполнение от начала и до конца.

Листинг 7.9 Отображение и обновление wx.Gauge

```
import wx

class GaugeFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Gauge Example',
                size=(350, 150))
        panel = wx.Panel(self, -1)
        self.count = 0
        self.gauge = wx.Gauge(panel, -1, 50, (20, 50), (250, 25))
        self.gauge.SetBezelFace(3)
        self.gauge.SetShadowWidth(3)
        self.Bind(wx.EVT_IDLE, self.OnIdle)

    def OnIdle(self, event):
        self.count = self.count + 1
        if self.count >= 50:
            self.count = 0
        self.gauge.SetValue(self.count)

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

Конструктор виджета wx.Gauge аналогичен другим числовым виджетам:

```
wx.Gauge(parent, id, range, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=wx.GA_HORIZONTAL,
        validator=wx.DefaultValidator, name="gauge")
```

Когда Вы с помощью аргумента range вводите числовое значение, оно представляет собой верхний предел индикатора, тогда как нижний предел всегда равен 0. Стиль по умолчанию wx.GA_HORIZONTAL реализует горизонтальный индиватор прогресса. Чтобы повернуть его на 90 градусов, используйте стиль wx.GA_VERTICAL. Если вы имеете дело с Windows, стиль wx.GA_PROGRESSBAR дает Вам стандартный индикатор прогресса из пакета разработчика Windows.

Виджет `wx.Gauge` не имеет событий, так как является элементом управления, работающим только в режиме чтения. Тем не менее, он имеет свойства, которые Вы можете устанавливать. Вы можете отрегулировать его значение и диапазон при помощи методов `GetValue()`, `SetValue(pos)`, `GetRange()` и `SetRange(range)`. Если вы имеете дело с Windows и не используете стиль стандартного индикатора прогресса, то для изменения ширины 3D-эффекта Вы можете использовать методы `SetBezelFace(width)` и `SetShadowWidth()`.

7.4 Предоставление пользователю возможности выбора

Практически каждое приложение в какой-то момент требует от пользователя сделать выбор среди предложенного набора альтернатив. В wxPython имеется ряд виджетов, которые помогают пользователю в этой задаче, среди них: флажки (checkboxes), переключатели (radio buttons), списки (list boxes) и комбинированные списки (combo boxes). Следующий раздел познакомит Вас с этими виджетами.

7.4.1 Как создать флажок (checkbox)?

Флажок (checkbox) – это кнопка-переключатель с текстовой меткой. Флажки часто выводятся группами, но каждый флажок имеет независимое состояние. Флажки используются, когда у вас есть одна или несколько опций, имеющих явные состояния «включено/выключено», и состояние отдельной опции не влияет на состояние других. Рисунок 7.10 отображает группу флажков.

Рисунок 7.10 Группа флажков wxPython

Флажки в wxPython использовать очень легко. Они являются экземплярами класса `wx.CheckBox` и могут отображаться вместе при их совместном размещении внутри родительского контейнера. Листинг 7.10 содержит код, который генерирует рисунок 7.10.

Листинг 7.10 Вставка во фрейм трёх флажков

```
import wx

class CheckBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Checkbox Example',
                size=(150, 200))
        panel = wx.Panel(self, -1)
        wx.CheckBox(panel, -1, "Alpha", (35, 40), (150, 20))
        wx.CheckBox(panel, -1, "Beta", (35, 60), (150, 20))
        wx.CheckBox(panel, -1, "Gamma", (35, 80), (150, 20))

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

Класс wx.CheckBox имеет типичный конструктор wxPython:

```
wx.CheckBox(parent, id, label, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=0, name="checkBox")
```

Аргумент `label` принимает отображаемый около флажка текст. Флажки не имеют свойственных только им стилевых флагов, однако они инициируют уникальное командное событие `EVT_CHECKBOX`. Состояние флажка `wx.CheckBox` может быть получено методами `GetValue()` и `SetValue(state)`, и оно имеет логическое значение Boolean. Метод `IsChecked()` идентичен методу `GetValue()` и реализован, для того чтобы сделать код более понятным.

7.4.2 Как создать группу переключателей (radio button)?

Рисунок 7.11 Пример wx.RadioButton, где переключатели включают текстовые поля

Переключатель (radio button) является виджетом, который позволяет пользователю выполнить выбор из нескольких опций. В отличие от флажков, переключатели явно помещены в группы и за один раз может быть выбрана только одна опция. Когда выбирается новая опция, действующая опция выключается. Альтернативное название переключателя – «радио-кнопка» происходит от группы кнопок в старых автомобильных радиоприёмниках, которые ведут себя тем же образом. В использовании переключатели немного сложнее, чем флажки, поскольку для их применения они должны быть сгруппированы.

В wxPython имеется два способа создать группу переключателей. Один из них – виджет `wx.RadioButton` требует, чтобы Вы создали кнопки поочередно, тогда как другой – виджет `wx.RadioBox` позволяет Вам развернуть целую группу кнопок в рамках единого объекта, отображающего кнопки в прямоугольнике.

Класс `wx.RadioButton` проще и предпочтительнее, когда переключатели непосредственно воздействуют на другие виджеты, или когда переключатели размещаются вне простого прямоугольника. Рисунок 7.11 отображает пример нескольких сгруппированных объектов `wx.RadioButton`.

Мы использовали в этом примере `wx.RadioButton`, поскольку каждый переключатель управляет связанным с ним текстовым полем. Мы не можем использовать группу переключателей, поскольку привлеченные виджеты находящиеся за пределами группы.

Как создать переключатель

Листинг 7.11 содержит код для рисунка 7.11, который управляет взаимодействием между перключателями и текстовыми элементами управления.

Листинг 7.11 Использование `wx.RadioButton` для управления другими виджетами

```
import wx
```

```
class RadioButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Radio Example',
                size=(200, 200))
        panel = wx.Panel(self, -1)
        radio1 = wx.RadioButton(panel, -1, "Elmo", pos=(20, 50),
          style=wx.RB_GROUP)
        radio2 = wx.RadioButton(panel, -1, "Ernie", pos=(20, 80))
        radio3 = wx.RadioButton(panel, -1, "Bert", pos=(20, 110))
        text1 = wx.TextCtrl(panel, -1, "", pos=(80, 50))
        text2 = wx.TextCtrl(panel, -1, "", pos=(80, 80))
        text3 = wx.TextCtrl(panel, -1, "", pos=(80, 110))
        self.texts = {"Elmo": text1, "Ernie": text2, "Bert": text3}
        for eachText in [text2, text3]:
            eachText.Enable(False)
        for eachRadio in [radio1, radio2, radio3]:
            self.Bind(wx.EVT_RADIOBUTTON, self.OnRadio, eachRadio)
        self.selectedText = text1

    def OnRadio(self, event):
        if self.selectedText:
            self.selectedText.Enable(False)
        radioSelected = event.GetEventObject()
        text = self.texts[radioSelected.GetLabel()]
        text.Enable(True)
        self.selectedText = text

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

Создание переключателя

Создание текстовых полей

Связывание кнопок и текста

Присоединение событий

Обработчик события

Мы создали переключатели и текстовые поля, затем инициализировали словарь, содержащий связи между ними. Первый цикл `for` отключает два текстовых поля, а второй связывает с переключателем командное событие. Когда событие происходит, текущее активное текстовое поле отключается и включается поле, соответствующее нажатому переключателю.

Виджет `wx.RadioButton` используется аналогично виджету `wx.CheckBox`. Как показано ниже, их конструкторы практически идентичны:

```
wx.RadioButton(parent, id, label, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=0,
        validator=wx.DefaultValidator, name="radioButton")
```

Как и у флажка, параметр `label` используется в качестве отображаемой рядом с переключателем текстовой метки.

Стиль `wx.RB_GROUP` определяет, что данная кнопка становится началом новой группы переключателей. Определение группы переключателей важно, поскольку оно управляет их поведением. Когда в группе выбирается одна кнопка, ранее выбранная кнопка этой группы переключается в неотмеченное состояние. После того, как будет создан переключатель с флагом `wx.RB_GROUP`, все последующие

добавляемые к тому же родителю переключатели добавляются в эту же группу. Это продолжается до тех пор, пока не будет создан другой переключатель с флагом `wx.RB_GROUP`, начинающий следующую группу переключателей. В листинге 7.11 первый переключатель объявлен с флагом `wx.RB_GROUP`, а последующие без него. В результате все кнопки будут в одной группе, и щелчок на одной из них отключает ранее выбранную кнопку.

Использование панели переключателей

Обычно, когда Вам необходимо показать группу переключателей, их раздельное объявление не является лучшим методом. Вместо этого, класс wxPython `wx.RadioBox`, позволяет Вам создать единый объект, объединяющий целую группу кнопок. Как показано на рисунке 7.12, он очень похож на группу переключателей.

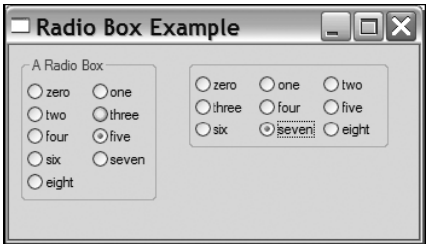

Рисунок 7.12
Два примера wx.RadioBox, созданных на одних данных в различных конфигурациях

Чтобы использовать класс `wx.RadioBox`, все что Вам необходимо – это конструктор. Листинг 7.12 содержит код, создавший рисунок 7.12.

Листинг 7.12 Построение панели переключателей

```
import wx

class RadioBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Radio Box Example',
                size=(350, 200))
        panel = wx.Panel(self, -1)
        sampleList = ['zero', 'one', 'two', 'three', 'four', 'five',
                      'six', 'seven', 'eight']
        wx.RadioBox(panel, -1,
                "A Radio Box", (10, 10), wx.DefaultSize,
                sampleList, 2, wx.RA_SPECIFY_COLS)

        wx.RadioBox(panel, -1, "", (150, 10), wx.DefaultSize,
                sampleList, 3, wx.RA_SPECIFY_COLS)

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

Конструктор `wx.RadioBox` сложнее, чем у простого переключателя, так как Вам необходимо определить данные сразу для всех кнопок:

```
wx.RadioBox(parent, id, label, pos=wx.DefaultPosition,
```

```
        size=wxDefaultSize, choices=None, majorDimension=0,
        style=wx.RA_SPECIFY_COLS, validator=wx.DefaultValidator,
        name="radioBox")
```

Данный конструктор имеет несколько аргументов, с которыми Вы еще незнакомы. Его аргумент `label` является отображаемым на границе панели статическим текстом. Сами кнопки определены аргументом choices, который является списком строк Python.

Подобно сеточному координатору, Вы определяете размерность `wx.RadioBox`, устанавливая размер в одном измерении, а wxPython занимает в другом измерении столько посадочных мест, сколько потребуется. Основной размер измерения определяется аргументом `majorDimension`. Какое измерение принимается в качестве основного, определяет стилевой флаг. Значение этого флага по умолчанию, использованное также в примере из листинга 7.12 - `wx.RA_SPECIFY_COLS`. В примере количество столбцов установлено в 2 (для левой панели) и 3 (для правой панели), а количество строк определяется динамически количеством элементов в списке `choices`. Если Вы хотите противоположное поведение, установите стиль в `wx.RA_SPECIFY_ROWS`. Если Вы хотите реагировать на командное событие при щелчке на панели переключателей, используйте событие `EVT_RADIOBOX`.

Класс `wx.RadioBox` имеет множество методов для управления состоянием отдельных кнопок панели переключателей. Методам, которые позволяют Вам управлять отдельной внутренней кнопкой, передается индекс этой кнопки. Индексы начинаются с 0 и действуют в том же порядке, в котором метки кнопок были переданы конструктору. Таблица 7.11 приводит эти методы.

Таблица 7.11 Методы `wx.RadioBox`

Метод	Описание
`EnableItem(n, flag)` `Enable()`	Аргумент `flag` типа Boolean используется для блокирования или разблокирования кнопки с индексом n. Для разблокирования сразу всей панели используется метод `Enable()`.
`FindString(string)`	Возвращает целый индекс кнопки с указанной в `string` меткой или значение -1, если метка не найдена.
`GetCount()`	Возвращает количество кнопок в панели.
`GetItemLabel(n)` `SetItemLabel(n, string)`	Возвращает или устанавливает строку метки для кнопки с индексом n.

`GetSelection()`
`GetStringSelection()`
`SetSelection(n)`
`SetStringSelection(string)`
	Методы `GetSelection()` и `SetSelection()` управляют целым индексом текущей выбранной кнопки переключателя. `GetStringSelection()` возвращает строку метки текущей выбранной кнопки, а `SetStringSelection()` изменяет метку выбранной кнопки данной строкой string. Ни один из установочных методов не вырабатывает событие `EVT_RADIOBOX`.

`ShowItem(item, show)`	Аргумент `show` типа Boolean используется для того, чтобы показать или скрыть кнопку с индексом `item`.

Переключатели предлагают не единственный способ предоставить пользователю выбор из набора опций. Простые и *комбинированные* списки зачастую занимают меньшее пространство и могут быть также сконфигурированы, чтобы предоставить пользователю возможность выполнить множественный выбор в той же группе опций.

7.4.3 Как создать список (listbox)?

Список (listbox) является иным механизм поддержки выбора пользователя. Опции размещаются в прямоугольном окне, и пользователь может выбрать одну или несколько из них. Списки занимают меньше места, чем переключатели и являются хорошим предпочтением при сравнительно небольшом количестве опций. Тем не менее, их полезность отчасти снижается в случае списка большого объема, когда для того чтобы увидеть все опции, пользователю необходимо пролистывать весь список. Рисунок 7.13 показывает список wxPython.

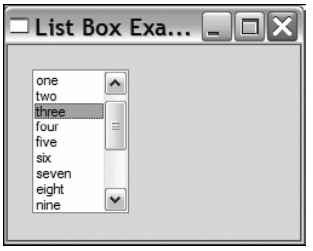

Рисунок 7.13 Виджет `wx.ListBox` с простым списком опций

Список wxPython является элементом класса `wx.ListBox`. Этот класс имеет методы, которые позволяют Вам управлять выбором в списке.

Создание списка

Листинг 7.13 показывает код создания списка для рисунка 7.13.

Листинг 7.13 Использование `wx.ListBox`

```
import wx

class ListBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'List Box Example',
                size=(250, 200))
        panel = wx.Panel(self, -1)
```

```
        sampleList = ['zero', 'one', 'two', 'three', 'four', 'five',
                      'six', 'seven', 'eight', 'nine', 'ten', 'eleven',
                      'twelve', 'thirteen', 'fourteen']

        listBox = wx.ListBox(panel, -1, (20, 20), (80, 120),
                sampleList, wx.LB_SINGLE)
        listBox.SetSelection(3)

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

Как видно ниже, конструктор `wx.ListBox` похож на конструктор переключателя:

```
wx.ListBox(parent, id, pos=wx.DefaultPosition,
        size=wx.DefaultSize, choices=None, style=0,
        validator=wx.DefaultValidator, name="listBox")
```

Основное отличие переключателя от списка состоит в том, что `wx.ListBox` не имеет атрибута `label`. Элементы, которые необходимо отображать в списке устанавливаются с помощью аргумента `choices`, который должен быть последовательностью строк. Из таблицы 7.12 видно, что имеется три взаимно исключающих стиля, которые определяют то, как пользователь выбирает элементы списка.

Поскольку пользователи обычно расчитывают увидеть список, допускающий единичный выбор, они часто испытывают проблемы со списками, допускающими множественный и расширенный выбор, к тому же поддержка множественного выбора может вызвать трудности, особенно у пользователей с ограниченными возможностями. Если Вы применяете список, допускающий множественный или расширенный выбор, мы рекомендуем, чтобы такой список Вы явно выделяли.

Таблица 7.12 Стили списка, задающие тип выбора

Стиль	Описание
`wx.LB_EXTENDED`	Пользователь может выбрать диапазон в множестве элементов списка при помощи совмещенного с клавишей Shift щелчка мыши или соответствующего клавиатурного эквивалента.
`wx.LB_MULTIPLE`	Пользователь может выбрать за один раз более одного элемента списка. По существу, в этом случае, список действует подобно группе флажков.
`wx.LB_SINGLE`	Пользователь может выбрать за один раз только один элемент списка. По существу, в этом случае, список действует подобно группе переключателей.

К тому же, как показано в таблице 7.13, имеется три стиля, которые управляют отображением в wx.ListBox полос прокрутки.

Таблица 7.13 Стили списка, задающие тип полос прокрутки

Стиль	Описание
wx.LB_ALWAYS_SB	Список будет всегда отображать вертикальную полосу прокрутки, не зависимо от того, нужна она или нет.
wx.LB_HSCROLL	Если этот флаг поддерживается виджетом, список создает горизонтальную полосу прокрутки, когда элементы списка не вписываются в его ширину.
wx.LB_NEEDED_SB	Список будет при необходимости отображать вертикальную полосу прокрутки. Этот флаг установлен по умолчанию.

Имеется также стиль wx.LB_SORT, указывающий на то, что элементы списка должны быть отсортированы в алфавитном порядке.

У wx.ListBox имеется два специальных командных события. Событие EVT_LISTBOX инициируется, когда выбирается элемент списка (даже, если он в настоящий момент является выбранным элементом). Если на списке производится двойной щелчок мышью, происходит событие EVT_LISTBOX_DCLICK.

Есть несколько методов, специально предназначенных для манипулирования элементами списка. Большинство из них описывает таблица 7.14. Все индексы начинаются с нуля и представляют текущий список элементов списка сверху вниз.

Как только у Вас появляется список, Вы естественно захотите объединить его с другими виджетами, например, с ниспадающим меню или флажками. В следующем разделе мы изучим эти возможности.

Таблица 7.14 Методы списков

Метод	Описание
Append(item)	Добавляет строку item в конец списка.
Clear()	Очищает список.
Delete(n)	Удаляет из списка элемент с индексом n.
Deselect(n)	В допускающих множественный выбор списках приводит к снятию отметки с элемента в позиции n. Не имеет действия для других стилей списка.
FindString(string)	Возвращает целочисленную позицию строки string или -1, если строка не найдена.
GetCount()	Возвращает количество строк списка.

`GetSelection()`
`SetSelection(n, select)`
`GetStringSelection()`
`SetStringSelection(string, select)`
`GetSelections()`
	Метод `GetSelection()`возвращает целочисленный индекс текущего выбранного элемента списка (только для списка с одиночным выбором). Для списка с множественным выбором используется метод `GetSelections()`, он возвращает кортеж целочисленных позиций. Для списка с одиночным выбором метод `GetStringSelection()` возвращает строку в позиции выбранного элемента.
	Методы установки `Set` устанавливают элемент списка, заданный индексом `n` или строкой `string` в состояние, определенное булевым аргументом `select`. Изменение выбранного элемента таким способом не инициирует событие `EVT_LISTBOX`.

`GetString(n)` `SetString(n, string)`	Возвращает или устанавливает строку `string` в позиции `n`.
`InsertItems(items, pos)`	Вставляет в список непосредственно перед заданной аргументом `pos` позицией список строк, указанный в аргументе `items`. Если `pos` равен 0, элементы будут добавлены в начало списка.
`Selected(n)`	Возвращает булевое значение состояния элемента списка с индексом `n`.
`Set(choices)`	Замещает список указанным в аргументе `choices` списком, т. е. все текущие элементы списка удаляются и в нем размещаются новые элементы.

7.4.4 Как комбинировать список с флажками?

Вы можете объединить список с флажками при помощи класса `wx.CheckListBox`. Такое объединение показано на рисунке 7.14.

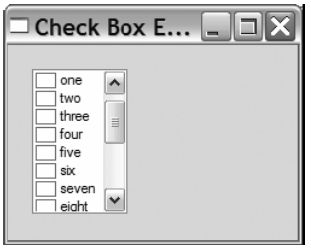

Рисунок 7.14
Список флажков очень похож на простой список

Конструктор и большинство методов `wx.CheckListBox` идентичны классу `wx.ListBox`. Имеется одно новое событие, `wx.EVT_CHECKLISTBOX`, которое происходит при щелчке на одном из флажков списка. Для управления флажками имеется два новых метода: `Check(n, check)` устанавливает флажок с индексом `n` в состояние `check`, а `IsChecked(item)` возвращает True, если флажок с указанным индексом отмечен.

7.4.5 Что если необходим выпадающий список (pull-down choice)?

Выпадающий список предоставляет механизм выбора, который показывает опции только когда щелкается стрела разворота списка. Разворачивание - наиболее компактный способ отобразить выбираемые элементы и исключительно полезен при ограниченном экранном пространстве. С точки зрения пользователей, такой

выбор полезен при относительно небольшом списке опций, с другой же стороны нет необходимости в любое время видеть все опции. На рисунке 7.15 показан закрытый список, а на рисунке 7.16 он находится в развернутом состоянии.

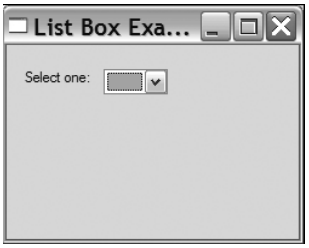

Рисунок 7.15 Выпадающий список, не имеющий выбранного значения

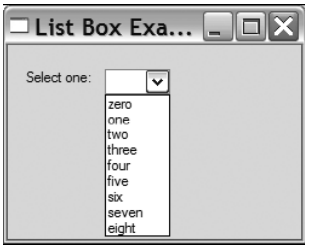

Рисунок 7.16 Выпадающий список в процессе выбора элемента

Выполнение выбора в таком списке очень похоже на обычный список. Листинг 7.14 показывает, как создать выпадающий список.

Листинг 7.14 Создание выпадающего списка

```
import wx

class ChoiceFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Choice Example',
                size=(250, 200))
        panel = wx.Panel(self, -1)
        sampleList = ['zero', 'one', 'two', 'three', 'four', 'five',
                      'six', 'seven', 'eight']
        wx.StaticText(panel, -1, "Select one:", (15, 20))
        wx.Choice(panel, -1, (85, 18), choices=sampleList)

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

Конструктор выпадающего списка практически идентичен обычному списку:

```
wx.Choice(parent, id, pos=wx.DefaultPosition,
        size=wx.DefaultSize, choices=None, style=0,
        validator=wx.DefaultValidator, name="choice")
```

Класс `wx.Choice` не имеет стилей, но имеет уникальное командное событие `EVT_CHOICE`. Почти все методы из таблицы 7.14, относящиеся к спискам, допускающим единственный выбор, также применимы и к объектам класса `wx.Choice`.

7.4.6 Как комбинировать ввод текста и список (combo box)?

Виджет, сочитающий ввод текста и список называется *комбинированным списком* (combo box), он представляет собой объединенное с выпадающим списком текстовое поле. Рисунок 7.17 показывает пример комбинированного списка.

В Windows Вы можете использовать его правосторонний стиль в виде соединенного со списком текстового поля.

Код для создания комбинированного списка аналогичен рассмотренному ранее выпадающему списку. В данном случае класс комбинированного списка `wx.ComboBox` является прямым подклассом `wx.Choice`. Листинг 7.15 показывает особенности его кода.

Листинг 7.15 Демонстрация `wx.ComboBox`

```
import wx

class ComboBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Combo Box Example',
                size=(350, 300))
        panel = wx.Panel(self, -1)
        sampleList = ['zero', 'one', 'two', 'three', 'four', 'five',
                      'six', 'seven', 'eight']
        wx.StaticText(panel, -1, "Select one:", (15, 15))
        wx.ComboBox(panel, -1, "default value", (15, 30),
                    wx.DefaultSize, sampleList, wx.CB_DROPDOWN)
        wx.ComboBox(panel, -1, "default value", (150, 30),
                    wx.DefaultSize, sampleList, wx.CB_SIMPLE)

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

Конструктор `wx.ComboBox` должен быть Вам понятен:

```
wx.ComboBox(parent, id, value="", pos=wx.DefaultPosition,
        size=wx.DefaultSize, choices, style=0,
        validator=wx.DefaultValidator, name="comboBox")
```

Класс `wx.ComboBox` имеет четыре стиля. Два из них определяют вид комбинированного списка: `wx.CB_DROPDOWN` создает комбинированный список с выпадающим списком, а `wx.CB_SIMPLE` создает комбинированный список с полностью раскрытым списком. Вы можете использовать `wx.CB_SIMPLE` только в системах Windows. Любой комбинированный список может быть определен со стилем `wx.CB_READONLY`, который не даёт пользователю вносить изменения в текстовое поле. Когда комбинированный список определен только для чтения, выбор должен поступать от одного из элементов списка, даже если Вы устанавливаете его программно. И, наконец, флаг `wx.CB_SORT` заставляет элементы списка отображаться в алфавитном порядке.

Так как wx.ComboBox это подкласс wx.Choice, комбинированный список может вызывать все приведенные в таблице 7.14 методы класса wx.Choice. Кроме того, определен набор методов для манипулирования текстовым полем, работающих так же, как и в объектах wx.TextCtrl (дополнительные сведения смотрите в таблице 7.4). Этот набор включает следующие методы: Copy(), Cut(), GetInsertionPoint(), GetValue(), Paste(), Replace(from, to, text), Remove(from, to), SetInsertionPoint(pos), SetInsertionPointEnd() и SetValue().

7.5 Резюме

В этой главе мы показывали Вам, как использовать большинство основных часто используемых элементов управления wxPython. Общая версия этих элементов является в определенной степени наиболее совместимой на разных платформах.

- Для отображения надписей статического текста Вы можете использовать класс wx.StaticText. Есть также версия, реализованная исключительно на wxPython – это класс wx.lib.stattext.GenStaticText.
- Если Вам нужен элемент управления, позволяющий пользователю вводить текст, используйте класс wx.TextCtrl. Он допускает как однострочный, так и многострочный ввод текста, а также маскирование пароля и другие эффекты. Если это поддерживается виджетом, Вы можете использовать wx.TextCtrl для стилизации текста. Стили являются экземплярами класса wx.TextAttr, который к тому же использует класс wx.Font для включения информации о шрифте. На всех системах Вы можете использовать класс wx.stc.StyledTextCtrl, являющийся wxPython-надстройкой к текстовому компоненту с открытым исходным кодом Scintilla, которая позволяет добиться оформления редактируемого текстового компонента на основе цветовых и шрифтовых стилей.
- Для создания кнопки используется класс wx.Button, который имеет также типового двойника - wx.lib.buttons.GenButton. Вместо текстовой надписи кнопка может иметь битовое изображение (wx.BitmapButton) или переключаться между нажатым и не нажатым состояниями (wx.ToggleButton). Существуют типовые эквиваленты кнопки с битовым изображением и кнопки-переключателя, которые имеют полный диапазон возможностей, что и у стандартных версий.
- Имеется несколько способов выбора или отображения числовых значений. Вы можете использовать класс wx.Slider для вывода вертикального или горизонтального ползунка. Виджет wx.SpinCtrl отображает текстовое поле с кнопками в виде стрелок вверх и вниз, и применяется для изменения числового значения. Виджет wx.Gauge служит для отображения числового индикатора прогресса.

- Для предоставления пользователю возможности выбора из списка опций у Вас имеется целая серия элементов управления. Решение о выборе того или иного элемента управления основывается на количестве опций, на том, может ли пользователь выбирать более чем одну опцию, а также на том, какое экранное пространство Вам необходимо. Флажки управляются классом `wx.CheckBox`. Есть два способа получить переключатели: `wx.RadioButton` реализует одиночный переключатель, а `wx.RadioBox` дает целую группу кнопок, отображаемых вместе. Имеется несколько используемых сходным образом виджетов для отображения списков. Простой список создает класс `wx.ListBox`, а, используя `wx.CheckListBox`, Вы можете разместить в списке флажки. Для более компактного способа отображения списка используется выпадающий список `wx.Choice`. Класс `wx.ComboBox` объединяет возможности списка и текстового поля.

Теперь, когда мы рассмотрели основы общих виджетов, в следующей главе мы обсудим различные виды фреймов, которые Вы можете использовать для размещения своих виджетов.