wxPython in Action
NOEL RAPPIN, ROBIN DUNN

Перевод: Володеев Сергей

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

Глава 11. Размещение виджетов с sizers

Эта глава включает

- Понимание sizers
- Размещение виджетов в sizers
- Использование grid sizers
- Использование box sizers
- Sizers в действии

Традиционно, одна из наиболее раздражающих проблем в программировании интерфейса - управление физическим размещением виджетов в пределах окна. В начале, было абсолютное позиционирование, и программист явно устанавливал размер и позицию каждого виджета на экране. Процесс мягко выражаясь утомительный, не говоря уже о боли в шее. Хуже того, абсолютное позиционирование заставляет вас постоянно контролировать размер окна и расположение виджетов. Если вы позволяете пользователю, изменять размер окна, то вы должны обработать это событие, и явно изменить размер и позицию каждого виджета. После этого удостоверьтесь, что виджеты все еще выглядят хорошо, не накладываются друг на друга и не вылезают за край окна. Это даже хуже боли в шее. Ситуация еще более ухудшается, если пользователь может изменять число и тип виджетов на экране. И, конечно, если вы или ваш клиент решаете, что нужно внести изменения в интерфейс, вы должны пройти этот процесс снова и снова.

Нужна структура, которая решает, как изменить размеры и переместить виджеты, основанная на предопределенном шаблоне. Было предложено несколько решений этой проблемы. Рекомендованный способ работы со сложным размещением виджетов - использование sizers. Sizer содержит автоматизированный алгоритм для размещения группы виджетов. Sizer связан с контейнером, обычно с фреймом или панелью. Виджеты, которые созданы в пределах родительского контейнера, должны также быть добавлены в sizer. Когда sizer связан с контейнером, именно он управляет размещением элементов.

Преимущества использования sizers существенны. Sizer автоматически обрабатывает событие изменения размера контейнера, повторно вычисляя размещение его виджетов. С другой стороны, если изменились размеры одного из виджетов, sizer может автоматически обновить размещение. Кроме того, sizers просты в использовании, когда вы хотите изменить размещение. Использование sizers наклкдывает определенные ограничения на размещение объектов. Однако, самые гибкие sizers - grid bag и box - будут в состоянии сделать почти все, что вы захотите.

11.1 Что такое - sizer?

В wxPython sizer – это объект, единственная цель которого состоит в том, чтобы управлять размещением виджетов в пределах контейнера. Sizer не контейнер и не виджет. Он только представление алгоритма размещения объектов на экране. Все sizers - объекты классов, производных от абстрактного класса wx.Sizer. Есть пять sizers представленных в wxPython. Они перечислены в таблице 11.1. Помните, sizer могжет быть вложен в другой sizer, чтобы предоставить вам больше гибкости.

Таблица 11.1 Предопределенные sizers в wxPython

Sizer	**Описание**
Grid	Один из основных способов размещения. Лучше всего использовать, когда виджеты имеют одинаковый размер и аккуратно попадают в ячейки сетки.
Flex grid	Немного отличается от grid sizer, позволяя получить лучшие результаты, когда виджеты имеют различные размеры.
Grid bag	Самый гибкий представитель семейства grid sizer. Используется для произвольного размещение виджетов в сетке.
Box	Горизонтальный или вертикальный бокс с виджетами, вытянутыми в линию. Очень гибко контролирует поведение виджетов при изменении размеров. Обычно вложен в другие sizers. Полезен почти для любого вида размещения.
Static box	Стандартный sizer с рамкой и заголовком.

Если вы хотите, чтобы ваше размещение было подобно сетке или боксу, wxPython определенно можно приспособить для этого; практически любое полезное размещение можно представить или как сетку или как ряд боксов.

Все sizers знают минимальный размер каждого из дочерних виджетов. Как правило, sizer также учитывает дополнительную информацию о размещении, например, промежутки между виджетами, или насколько можно увеличить размер виджета, и как выровнять виджеты, когда они занимают места меньше, чем выделено. Эту информацию использует алгоритм размещения, чтобы определить размер и позицию каждого виджета. Каждый вид sizers в wxPython произведет различное размещение той же самой группы виджетов. Вы увидите это далее, в этой главе, поскольку мы используем похожие размещения, чтобы демонстрировать каждый тип sizers.

Три основных шага для использования sizer:

1. Добавьте sizer к контейнеру. Подключите sizer к виджету, дочерними объектами которого он управляет. Sizer добавляется к контейнерному виджету, используя метод SetSizer(sizer) класса wx.Window. Так как это метод класса wx.Window, это означает, что любой виджет wxPython может иметь sizer, хотя sizer используется только для виджетов, которые являются контейнерами.

2. Добавьте каждый дочерний виджет к sizer. Все дочерние виджеты должны быть отдельно добавлены к sizer. Простого создания дочерних виджетов в родительском контейнере - недостаточно. Виджет добавляется к sizer методом Add(). Метод Add() имеет несколько различных сигнатур, которые будут рассмотрены в следующем разделе.

3. (дополнительный) Разрешить sizer вычислить его размер. Скажите sizer вычислить его размер, основанный на его дочерних виджетах, вызвав метод Fit() класса wx.Window на родительском объекте окна или метод Fit(window) sizer. (Метод окна переадресовывает к методу sizer.) В любом случае, метод Fit() просит, чтобы sizer вычислил его размер, основанный на дочерних виджетах, и изменяет размеры родительского виджета, чтобы соответствовать этому размеру. Есть связанный метод, FitInside(), который не изменяет размер отображения родительского виджета, но действительно изменяет его виртуальные размеры так, что, если бы виджет находился в прокручиваемой панели, wxPython повторно вычислил бы, действительно ли необходимы полосы прокрутки.

Далее мы должны рассмотреть поведение определенных видов sizers и поведение, общее для всех sizers. Всегда есть вопрос, с чего начать рассмотрение. Наше решение состоит в том, чтобы начать, с представления grid sizer, который является самым легким для

понимания. После этого, мы обсудим поведение, обычное для всех sizers, используя grid sizer как пример. Затем, мы покажем остальные типы sizers.

11.2 Grid sizer

Все следующие примеры используют простой виджет, цель которого состоит в том, чтобы занять место в размещении, таким образом вы можете видеть как работает sizer. В листинге 11.1 показан код для этого виджета, который импортируется остальными примерами этой главы. Это простой прямоугольник с надписью.

Листинг 11.1 Виджет, используемый в более поздних примерах

```
import wx

class BlockWindow(wx.Panel):
    def __init__(self, parent, ID=-1, label="",
                 pos=wx.DefaultPosition, size=(100, 25)):
        wx.Panel.__init__(self, parent, ID, pos, size,
                          wx.RAISED_BORDER, label)
        self.label = label
        self.SetBackgroundColour("white")
        self.SetMinSize(size)
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, evt):
        sz = self.GetClientSize()
        dc = wx.PaintDC(self)
        w,h = dc.GetTextExtent(self.label)
        dc.SetFont(self.GetFont())
        dc.DrawText(self.label, (sz.width-w)/2, (sz.height-h)/2)
```

Мы будем помещать несколько таких виджетов во фрейм, используя различные sizers далее в этой главе. Начнем с grid sizer.

11.2.1 Что такое - grid sizer?

Самый простой sizer, предлагаемый wxPython - grid. Название подразумевает, что grid sizer размещает виджеты в двумерной сетке. Первый виджет в списке дочерних объектов sizer помещается в левый верхний угол сетки, остальные размещаются слева направо и сверху вниз, пока последний виджет не займет правый нижний угол сетки. На рисунке 11.1 показан пример с девятью виджетами, помещенными в сетку 3 x 3. Заметьте, что есть небольшой промежуток между виджетами.

Рисунок 11.1

Когда вы изменяете размеры grid sizer, промежутки становятся больше, но по умолчанию размеры виджетов и расположение левого верхнего угла не изменяется. На рисунке 11.2 показано то же самое окно после изменений.

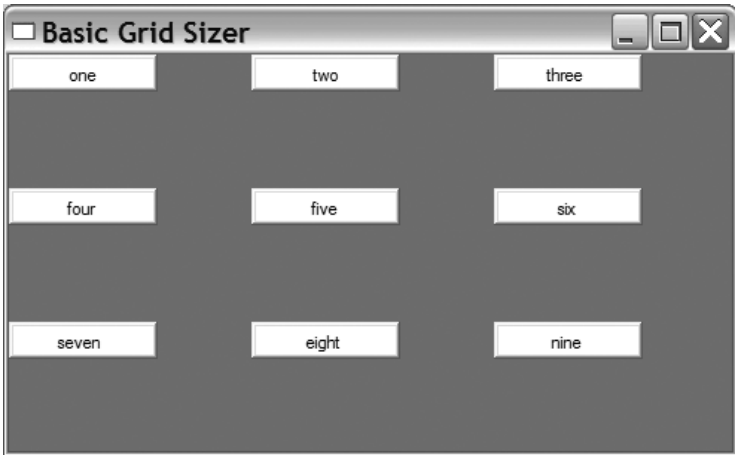

Рисунок 11.2

В листинге 11.2 показан код для рисунков 11.1 и 11.2.

Листинг 11.2 Использование grid sizer

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()

class GridSizerFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Basic Grid Sizer")
        sizer = wx.GridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            sizer.Add(bw, 0, 0)
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
GridSizerFrame().Show()
app.MainLoop()
```

Как видно из листинга 11.2, grid sizer - объект класса wx.GridSizer. Конструктор явно устанавливает четыре свойства, которые являются уникальными для grid sizer:

```
wx.GridSizer(rows, cols, vgap, hgap)
```

Параметры rows и cols - целые числа, которые определяют размер сетки — количество виджетов, которые будут помещены в строке и в столбце. Если какой-либо из параметров равен нулю, то значение другого параметра вычисляется на основе общего количества виджетов в sizer. Например, если sizer, созданный конструктором wx.GridSizer(2, 0, 0, 0) будет иметь восемь виджетов, то он должен будет иметь четыре столбца, чтобы соответствовать двум строкам.

Параметры vgap и hgap позволяют задать размер промежутков между виджетами. Параметр vgap - число пикселей между смежными столбцами, и hgap - число пикселей между смежными строками. Свойства rows, cols, vgap и hgap имеют соответствующие Set и Get методы: GetRows(), SetRows(rows), GetCols(), SetCols(cols), GetVGap(), SetVGap(gap), GetHGap() и SetHGap(gap).

Алгоритм установки размеров и размещения grid sizer является прямым. Он создает начальное размещение сетки, когда вызывает Fit(). В случае необходимости, число строк и столбцов вычисляется от числа элементов в списке. Каждая ячейка в сетке имеет одинаковый размер, даже если размеры виджетов различны. Она имеет размер самого большого дочернего виджета. Из-за этого, grid sizer лучше всего использовать для размещений, где виджеты имеют одинаковый размер (классический пример -

вспомогательная клавиатура калькулятора). Если grid sizer содержит виджеты, сильно различающиеся по размерам, то лучше использовать flex grid sizer или grid bag sizer.

11.2.2 Как добавить или удалить объект из sizer?

Порядок, в котором дочерние виджеты добавляются в sizer, очень важен. Это отличается от общего случая добавления дочерних объектов к родительскому виджету. Типичный алгоритм размещения для sizer берет каждый дочерний виджет по очереди, чтобы определить его место при отображении. Размещение следующего элемента зависитот того, как разместились предыдущие элементы. Например, grid sizer двигается слева направо и сверху вниз по списку виджетов. Обычно, если вы создаете sizer в родительском конструкторе виджета, вы будете в состоянии добавить элементы в правильном порядке. Однако, в некоторых случаях, вам будет нужна большая гибкость, особенно если вы будете динамически изменять размещение во время выполнения.

Использование метода Add()

Самый общий метод для добавления виджета к sizer – это метод Add(). Он добавляет виджет в конец дочернего списка sizer. Метод Add() имеет три различных стиля:

```
Add(window, proportion=0, flag=0, border=0, userData=None)
Add(sizer, proportion=0, flag=0, border=0, userData=None)
Add(size, proportion=0, flag=0, border=0, userData=None)
```

Первая версия используется чаще всего. Она позволяет добавлять виджет в sizer. Вторая версия используется, чтобы вложить один sizer в другой — обычно это делается с box sizers, но вы можете сделать это с любым типом sizer. Третья версия позволяет вам добавлять пустое пространство в sizer. Параметр size – это объекта wx.Size или кортеж (width, height). Пустое пространство используется как разделитель (например, в панели инструментов). Обычно это используется в box sizers, но может использоваться в любом sizer, чтобы добавить пустое пространство в область окна или разделить виджеты.

Другие параметры определяют, как элемент отображен в пределах sizer. Некоторые из этих параметров имеют значение только для определенного вида sizers. Элемент proportion используется только с box sizers, и указывает, насколько элемент деформируется, когда родительское окно изменяет размер. Это будет рассмотрено вместе с box sizers, позже в этой главе.

Параметр flag используется, чтобы поместить любой из битовых флагов, которые управляют выравниванием, границей и изменением размеров. Эти опции обсуждаются далее. Параметр border содержит ширину границы, если граница определена в опции flag. Параметр userData может использоваться, чтобы передать дополнительные данные sizer, если они нужны для его алгоритма. Вы могли бы использовать его, если бы проектировали собственный sizer.

Использование метода Insert()

Как вы могли бы ожидать, прочитав главу о Меню (глава 10), есть связанные методы, чтобы вставить новый виджет в sizer. Метод Insert() позволяет вам помещать новый виджет в список, используя произвольный индекс. Он также имеет три разновидности:

```
Insert(index, window, proportion=0, flag=0, border=0, userData=None)
Insert(index, sizer, proportion=0, flag=0, border=0, userData=None)
Insert(index, size, proportion=0, flag=0, border=0, userData=None)
```

Использование метода Prepend()

Есть также метод Prepend(), который добавляет новый виджет, sizer или разделитель в начало списка sizer:

```
Prepend(window, proportion=0, flag=0, border=0, userData=None)
Prepend(sizer, proportion=0, flag=0, border=0, userData=None)
Prepend(size, proportion=0, flag=0, border=0, userData=None)
```

Рисунок 11.3 показывает, на что было бы похоже размещение сетки из листинга 11.1, если бы вместо метода Add() использовался бы Prepend().

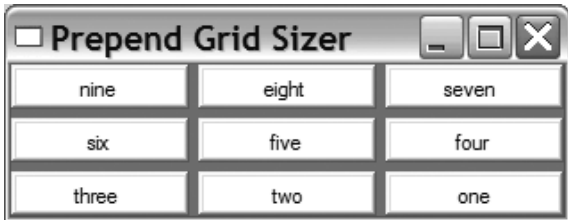

Рисунок 11.3

Если вы добавляете новые элементы к sizer после того, как он уже был отображен на экране, вы должны вызвать метод Layout(), чтобы заставить sizer перестроится.

Использование метода Detach()

Чтобы удалить элемент из sizer, используйте метод Detach(), который удаляет элемент из sizer, но не разрушает его. Это полезно, если вы хотите использовать этот элемент в будущем. Есть три способа использовать Detach(). Вы можете передать ему в качестве параметра окно, sizer, который вы хотите отделить, или вы можете передать целочисленный индекс объекта:

```
Detach(window)
Detach(sizer)
Detach(index)
```

Во всех трех случаях, метод Detach() возвращает булевское значение показывающее, был ли элемент фактически удален. Метод возвратит False, если вы пробуете удалить элемент, отсутствующий в sizer. В отличие от некоторых других методов удаления, которые мы рассматривали, Detach() не возвращает удаляемый элемент, поэтому, если вы хотите продолжить работу с этим объектом, вы должны уже иметь переменную, ссылающуюся на этот объект.

Удаление элемента из sizer не изменяет экран автоматически. Вы должны вызвать метод Layout(), чтобы обновить отображение.

Вы всегда можете получать ссылку на sizer, содержащий виджет, используя метод GetContainingSizer() класса wx.Window. Метод возвращает None, если виджет не содержится в sizer.

11.2.3 Как sizers управлют размером и выравниванием дочерних виджетов?

Когда новый элемент добавляется к sizer, sizer использует или начальный размер элемента или лучший размер элемента, если нет начального размера, в своих вычислениях размещения. Другими словами, sizer не корректирует размер элемента, пока к sizer не обращаются, обычно в контексте изменения размеров окна.

Когда родительский виджет sizer-а изменяется, sizer в ответ должен изменить размер своих компонентов. По умолчанию виджет в слоте выровнен по левому и по верхнему краям.

Вы можете корректировать поведение при изменении размеров каждого виджета, присваивая определенные значения параметру flag, когда вы добавляете виджет в sizer. На рисунке 11.4 показан результат действая нескольких различных флагов, после того, как пользователь увеличивает размер окна.

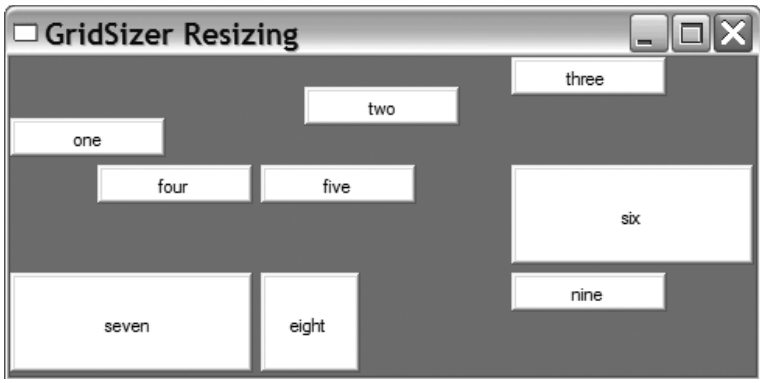

Рисунок 11.4

Листинг 11.3 содержит код для рисунка 11.4. Он идентичен предыдущему листингу, за исключением дополнительного словаря значений flags, которые будут применены к виджетам при добавлении. Эти значения показаны полужирным шрифтом.

Листинг 11.3 Grid sizer с флажками для выравнивания и установки размеров

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()
flags = {"one": wx.ALIGN_BOTTOM, "two": wx.ALIGN_CENTER,
         "four": wx.ALIGN_RIGHT, "six": wx.EXPAND, "seven": wx.EXPAND,
         "eight": wx.SHAPED}

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "GridSizer Resizing")
        sizer = wx.GridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            flag = flags.get(label, 0)
            sizer.Add(bw, 0, flag)
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

В этом примере, у виджетов “one,” “two,” и “four” изменен тип выравнивания использованием флагов wx.ALIGN_BOTTOM, wx.ALIGN_CENTER и wx.ALIGN_RIGHT, соответственно. Вы можете видеть, что поведение этих виджетов соответствует их типу выравнивания (поведение виджета “three” соответствует поведению по умолчанию). Виджеты “six” и “seven” используют флаг wx.EXPAND, заставляя sizer изменять их размер так, чтобы заполнить доступное пространство слота, в то время как виджет "eight" использует флаг wx.SHAPED, указывающий, что при изменении должны сохраняться его пропорции.

В таблице 11.2 перечислены возможные значения flag, которые можно использовать для установки размеров и выравнивания.

Таблица 11.2 Флаги размеров и выравнивания

Флаг	**Описание**
wx.ALIGN_BOTTOM	Выравнивает виджет по нижнему краю слота.
wx.ALIGN_CENTER	Размещает виджет так, чтобы центр виджета находился в центре слота.
wx.ALIGN_CENTER_HORIZONTAL	Размещает виджет в центре по горизонтали.
wx.ALIGN_CENTER_VERTICAL	Размещает виджет в центре по вертикали.
wx.ALIGN_LEFT	Выравнивает виджет по левому краю слота. Установлен по умолчанию.
wx.ALIGN_TOP	Выравнивает виджет по верхнему краю слота. Установлен по умолчанию.
wx.EXPAND	Изменяет размер виджета так, чтобы заполнить весь слот при любых изменениях размера родительского окна.
wx.FIXED_MINSIZE	Заставляет sizer сохранять минимальный размер элемента.
wx.GROW	То же самое что и wx.EXPAND.
wx.SHAPED	Изменяет размер виджета так, что виджет заполняет выделенное место по одному измерению, и другое измерение заполняется, сохраняя пропорции оригинальной формы виджета.

Так как флажки – это битовые маски, они могут быть объединены, используя операцию поразрядное ИЛИ (|), в случаях, где такая комбинация имела бы смысл. Так, wx.ALIGN_TOP | wx.ALIGN_RIGHT сохраняет виджет в правом верхнем углу слота. (Отметьте, что взаимно исключающие значения, типа wx.ALIGN_TOP | wx.ALIGN_BOTTOM будут всегда приводится к значению не по умолчанию, потому что значение по умолчанию - целое число 0. Оно не изменяет значение другого операнда в операции поразрядное ИЛИ.)

Есть несколько методов, которые вы можете использовать, чтобы управлять размером и позиционированием sizer или его составляющих виджетов во время выполнения. Вы можете получить текущий размер и позицию sizer, используя методы GetSize() и GetPosition() — позиция относительно контейнера, с которым связан sizer. Это является полезным, если sizer вложен в другой sizer. Вы можете принудительно задать размер sizer, вызвав метод SetDimension(x, y, width, height). После этого, sizer повторно вычислит размер своих дочерних виджетов.

11.2.4 Как задать минимальный размер для sizer и его дочерних виджетов?

Часто, вы не хотите, чтобы элемент управления или sizer стал меньше определенного размера. К счастью wxPython позволяет определить минимальный размер для sizer и его дочерних виджетов.

На рисунке 11.5 показан пример установки минимального размера для определенного виджета. Это окно не было изменено пользователем.

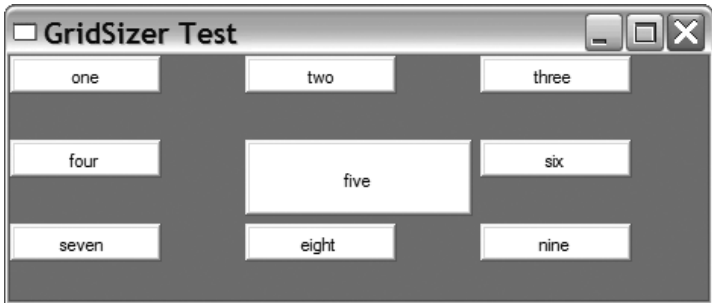

Рисунок 11.5

Листинг 11.4 показывает код для рисунка. Он подобен основному коду сетки с единственным дополнением - вызовом метода SetMinSize().

Листинг 11.4 grid sizer с установкой минимального размера

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()

class TestFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, "GridSizer Test")
        sizer = wx.GridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            sizer.Add(bw, 0, 0)
        center = self.FindWindowByName("five")
        center.SetMinSize((150,50))
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Когда sizer создан, он неявно задает минимальный размер, основанный на объединенном минимальном размере его виджетов. Минимальный размер элемента управления может также быть установлен, используя методы SetMinSize(width, height) и SetSizeHints(minW, minH, maxW, maxH). Последний метод также разрешает определить максимальный размер. Виджет будет обычно корректировать наилучший размер, если изменяются его атрибуты — например, тип шрифта или текст метки.

Вы можете получить минимальный размер для всего sizer, используя метод GetMinSize(). Если вы хотите установить больший минимальный размер sizer, используйте метод SetMinSize(width, height), который вы можете также вызвать с параметром wx.Size, хотя в wxPython редко явно создаются объекты wx.Size. После того, как минимальный размер установлен, метод GetMinSize() возвращает или установленный размер или объединенный размер дочерних виджетов, если он больше.

Если вы хотите только установить минимальный размер определенного дочернего виджета в пределах sizer, используйте метод SetItemMinSize() объекта sizer. Аналогично методу Detach(), есть три способа вызова SetItemMinSize(), с window, sizer или индексом:

```
SetItemMinSize(window, size)
SetItemMinSize(sizer, size)
SetItemMinSize(index, size)
```

В этом случае, window или sizer должны быть дочерними объектами sizer. Метод просматривает все вложенное дерево sizers для того, чтобы найти определенное окно или

sizer. Параметр index - индекс в списке элементов sizer. Параметр size - объект wx.Size или кортеж (width, height), задающий явный минимальный размер элемента в пределах sizer. Если минимальный размер, больше текущего размера виджета, он автоматически изменяется. Вы не можете установить максимальный размер методом sizer, его можно установить только методом виджета, используя SetSizeHints().

11.2.5 Как sizer управляет границей вокруг каждого элемента?

В wxPython sizer может поместить границу вокруг любого из своих элементов. Граница - пустое пространство, отделяющего виджет от его соседей. Размер границы учитывается, когда sizer вычисляет размещение своих объектов. Размер границы не изменяется, когда sizer изменяет размеры.

На рисунке 11.6 показана границе в 10 пикселей, помещенная вокруг всех или части виджетов в нашем основном размещении сетки. В каждой строке, средний элемент имеет границу вокруг всех четырех сторон, в то время как другие виджеты ограничены не со всех сторон. Добавление границы не делает виджеты меньшими; скорее это делает фрейм большим.

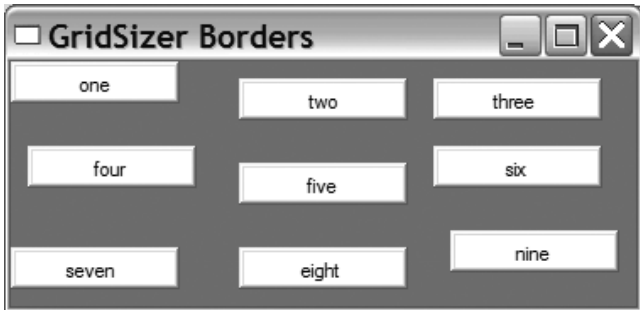

Рисунок 11.6

Листинг 11.5 содержит код для рисунка 11.6. Снова, он подобен основному grid sizer, но теперь мы добавили словарь значений границы и параметр размера границы в 10 пикселей в методе Add().

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()
flags = {"one": wx.BOTTOM, "two": wx.ALL, "three": wx.TOP,
         "four": wx.LEFT, "five": wx.ALL, "six": wx.RIGHT,
         "seven": wx.BOTTOM | wx.TOP, "eight": wx.ALL,
         "nine": wx.LEFT | wx.RIGHT}

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "GridSizer Borders")
        sizer = wx.GridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            flag = flags.get(label, 0)
            sizer.Add(bw, 0, flag, 10)
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Помещение границы вокруг виджета в sizer - двухступенчатый процесс. На первом шаге нужно передать дополнительные флажки в параметре flags при добавлении виджета в sizer. Вы можете определить границу вокруг всего виджета, используя флаг wx.ALL, или ограничить виджет с определенной стороны, используя wx.BOTTOM, wx.LEFT,

wx.RIGHT или wx.TOP. Естественно, флаги могут быть объединены для любого набора сторон, который вам нужен, например, флаг wx.RIGHT | wx.BOTTOM создаст границу справа и внизу виджета. Признаки границы, изменения размеров и выравнивания передают через тот же самый параметр flags. Вы часто будете использовать операцию поразрядное ИЛИ (|), чтобы объединить признаки границы с признаками установки размеров и выравнивания для виджета.

После того, как вы сформировали параметр flags, в следующем параметре вы должны передать ширину границы в пикселях. Например, следующая строка добавит виджет в конец списка sizer, помещая границу в пять пикселей вокруг всего виджета:

```
sizer.Add(widget, 0, wx.ALL | wx.EXPAND, 5)
```

При расширении виджет всегда будет оставлять вокруг себя пространство шириной в пять пикселей.

11.3 Использование других типов sizer

Рассмотрев основы, мы можем идти дальше к более сложным и гибким типам sizer. Два из них — flex grid sizer и grid bag sizer — являются по существу расширениями темы сетки. Другие два — box и static box sizers — используют различную и более гибкую структуру размещения.

11.3.1 Что такое - flex grid sizer?

Flex grid sizer - более гибкая версия grid sizer, которая почти идентична регулярному grid sizer, со следующими исключениями:

- Flex grid sizer определяет отдельный размер для каждой строки и столбца.
- По умолчанию, размеры ячеек не изменяются. Вы можете определить, какие строки или столбцы должны изменять размеры.
- Flex grid sizer может изменяться гибко в любом направлении.

Рисунок 11.7 показывает flex grid sizer в действии, используя то же самое размещение с девятью ячейками, которое используется для основного grid sizer. В этом случае, центральная ячейка была сделана большей.

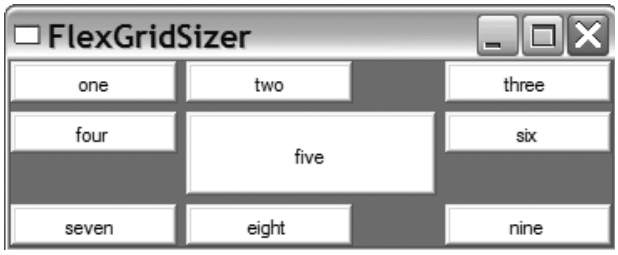

Рисунок 11.7

Сравните это изображение с рисунком 11.5, на котором изображено то же самое размещение в обычном grid sizer. В обычном grid sizer каждая ячейка имеет такой же размер как и средняя. В flex grid sizer, ячейки установлены по размеру согласно строке и столбцу, частью которых они являются. Они берут ширину самого широкого элемента в их столбце и высоту самого высокого элемента в их строке. В этом случае, ячейки для элементов “four” и “six” выше остальных, потому что они находятся в той же самой строке, что и элемент “five”, аналогично ячейки для элементов “two” и “seven” шире остальных. Ячейки для “one”, “three”, “seven” и “nine” - нормального размера, так как не затронуты большим виджетом.

На рисунке 11.8 показано поведению flex grid sizer по умолчанию при изменении размеров окна. Размер ячеек остается неизменным.

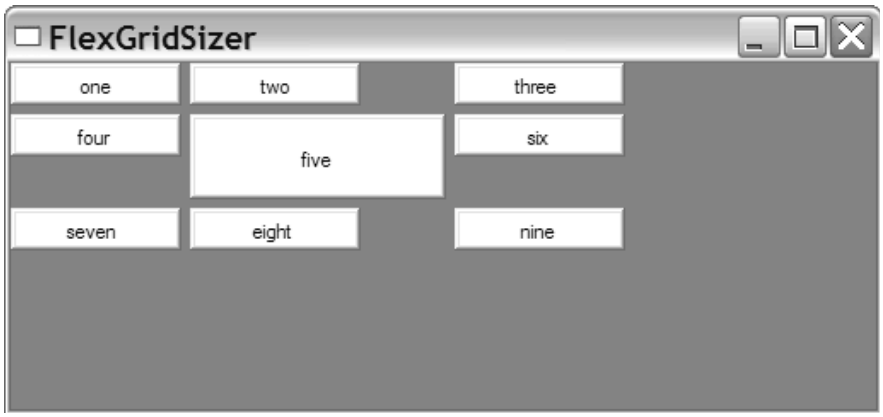

Рисунок 11.8

Листинг 11.6 содержит код для рисунка 11.8.

Листинг 11.6 Создание flex grid sizer

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "FlexGridSizer")
        sizer = wx.FlexGridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            sizer.Add(bw, 0, 0)
        center = self.FindWindowByName("five")
        center.SetMinSize((150,50))
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Flex grid sizer – объект класса wx.FlexGridSizer. Класс wx.FlexGridSizer является производным от wx.GridSizer, таким образом все методы и свойства wx.GridSizer доступны. Конструктор для wx.FlexGridSizer идентичен конструктору родительского класса:

```
wx.FlexGridSizer(rows, cols, vgap, hgap)
```

Чтобы размеры строки или столбца изменялись при изменении sizer, вы должны явно указать, что строка или столбец являются изменяемыми (growable), используя соответствующий метод:

```
AddGrowableCol(idx, proportion=0)
AddGrowableRow(idx, proportion=0)
```

Когда sizer увеличивается по горизонтали, новая ширина будет распределена одинаково между всеми изменяемыми столбцами. Это поведение по умолчанию. Аналогично, при изменении размера по вертикали, новая высота будет равномерно распределена между всеми изменяемыми строками. Чтобы изменить поведение по умолчанию и сделать так, чтобы строки или столбцы изменялись неравномерно, используйте параметр proportion. Если параметр proportion используется, то новое место будет распределено между строками или столбцами относительно их параметров пропорции. Например, если у вас

есть две строки, и их пропорции - 2 и 1, то тогда первая строка получит 2/3 нового места, а вторая строка получит 1/3. Рисунок 11.9 показывает flex grid sizer с пропорциональными интервалами. В этом случае, средняя строка и столбец имеют пропорцию 2, а строки и столбцы в конце имеют пропорцию 1. Рисунок 11.9 показывает на что это похоже.

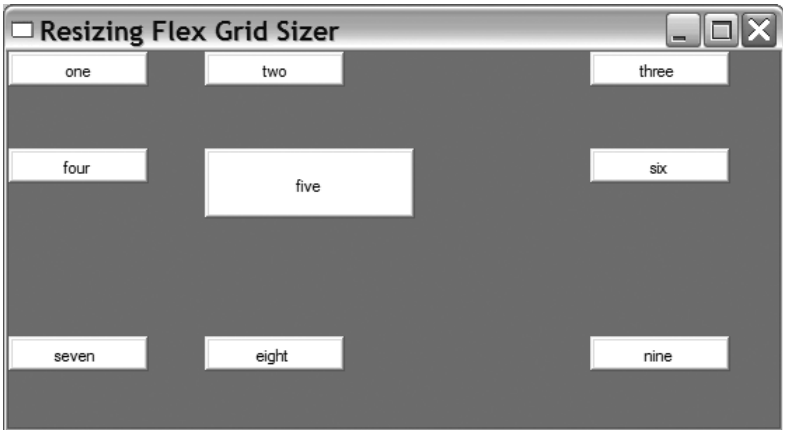

Рисунок 11.9

Как видите, в то время как все ячейки увеличились, средняя строка и столбец стали вдвое больше других строк и столбцов. Виджеты не изменили размеры, чтобы заполнить ячейки, хотя могли бы это сделать, если бы использовался флаг wx.EXPAND, когда они добавлялись к sizer. Листинг 11.7 показывает код для создания flex grid sizer. Обратите внимание на использование growable-методов.

Листинг 11.7 Flex grid sizer с изменяемыми (growable) элементами

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Resizing Flex Grid Sizer")
        sizer = wx.FlexGridSizer(rows=3, cols=3, hgap=5, vgap=5)
        for label in labels:
            bw = BlockWindow(self, label=label)
            sizer.Add(bw, 0, 0)
        center = self.FindWindowByName("five")
        center.SetMinSize((150,50))
        sizer.AddGrowableCol(0, 1)
        sizer.AddGrowableCol(1, 2)
        sizer.AddGrowableCol(2, 1)
        sizer.AddGrowableRow(0, 1)
        sizer.AddGrowableRow(1, 2)
        sizer.AddGrowableRow(2, 1)
        self.SetSizer(sizer)
        self.Fit()

app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Если вы используете пропорциональную установку размеров на одной из ваших изменяемых строк или столбцов, вы должны определить пропорции на всех изменяемых элементах на этом направлении, иначе строка или столбец с неустановленной пропорцией просто исчезнет при изменении размеров.

Есть еще один механизм, чтобы управлять изменением размеров ячейки в flex grid sizer. По умолчанию, пропорциональная установка размеров воздействует на оба направления flex grid (и по вертикали, и по горизонтали); однако, вы можете определить, что только

одно направление должно изменить размеры пропорционально, используя метод SetFlexibleDirection (direction), где значения direction - wx.HORIZONTAL, wx.VERTICAL или wx.BOTH (по умолчанию). Затем вы определяете поведение в другом направлении, используя SetNonFlexibleGrowMode(mode). Например, если вы вызываете SetFlexibleDirection (wx.HORIZONTAL), столбцы ведут себя как определено при использовании AddGrowableCol(), и вызов SetNonFlexibleGrowMode() определяет поведение строк. В таблице 11.3 показаны доступные значения для параметра mode.

Таблица 11.3 Non-flexible grow mode values

Режим	**Описание**
wx.FLEX_GROWMODE_ALL	Flex grid изменяет размеры всех ячеек в направлении non-flexible одинаково. Это отменяет любое поведение, заданное growable-методами — все ячейки изменяются независимо от их пропорции.
wx.FLEX_GROWMODE_NONE	Ячейки в направлении non-flexible не изменяются, независимо от того, были ли они определены как изменяемые.
wx.FLEX_GROWMODE_SPECIFIED	Sizer изменяет только те ячейки в направлении non-flexible, которые были определены как изменяемые. Однако, sizer игнорирует любую информацию о пропорции и изменяет все ячейки одинаково. Это - поведение по умолчанию.

Каждый из методов, обсуждаемых в предыдущем параграфе имеет связанный Get-метод, GetFlexibleDirection() и GetNonFlexibleGrowMode(), который возвращает целое число – значение флага.

11.3.2 Что такое - grid bag sizer?

Grid bag sizer - дальнейшее расширение flex grid sizer. В grid bag sizer есть два новшества.

- Способность добавлять виджет в заданную ячейку сетки.
- Способность иметь виджет, охватывающий несколько ячеек сетки (как ячейка в таблице HTML).

На рисунке 11.10 показан типичный grid bag sizer. Он подобен примеру, который мы использовали всюду в этой главе, с добавлением новых виджетов.

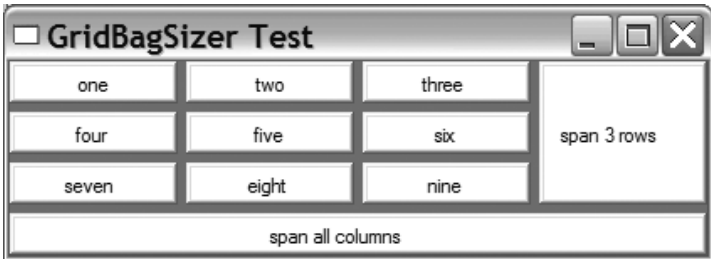

Рисунок 11.10

Листинг 11.8 содержит код для рисунка 11.10. Заметьте, что метод Add() выглядит немного иначе.

Листинг 11.8 Grid bag sizer

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()
```

```
class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "GridBagSizer Test")
        sizer = wx.GridBagSizer(hgap=5, vgap=5)
        for col in range(3):
            for row in range(3):
                bw = BlockWindow(self, label=labels[row*3 + col])
                sizer.Add(bw, pos=(row,col))

        # add a window that spans several rows
        bw = BlockWindow(self, label="span 3 rows")
        sizer.Add(bw, pos=(0,3), span=(3,1), flag=wx.EXPAND)

        # add a window that spans all columns
        bw = BlockWindow(self, label="span all columns")
        sizer.Add(bw, pos=(3,0), span=(1,4), flag=wx.EXPAND)

        # make the last row and col be stretchable
        sizer.AddGrowableCol(3)
        sizer.AddGrowableRow(3)

        self.SetSizer(sizer)
        self.Fit()


app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Grid bag sizer – объект класса wx.GridBagSizer, который является производным от wx.FlexGridSizer. Это означает, что можно использовать все методы flex grid sizer, включая добавление изменяемых строк и столбцов.

Конструктор для wx.GridBagSizer немного отличается от конструктора родительского класса:

```
wx.GridBagSizer(vgap=0, hgap=0)
```

Вы не должны определять количество строк и столбцов в grid bag sizer, потому что дочерние виджеты добавляются непосредственно в заданные ячейки сетки, а sizer сам вычисляет размерность сетки.

Использование метода Add() для grid bag sizer

Метод Add() для grid bag sizers отличается от аналогичного метода в других sizers. Он имеет четыре опции:

1. `Add(window, pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)`
2. `Add(sizer, pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)`
3. `Add(size, pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)`
4. `AddItem(item)`

Они выглядят знакомыми, будучи подобными методам базового класса. Параметры window, sizer, size, flag, border и userData ведут себя также, как в методе базового класса. Параметр pos представляет ячейку в пределах sizer, в которую помещается виджет. Технически, pos – это объект класса wx.GBPosition, но благодаря магии wxPython, вы можете передать кортеж (row, col). Верхняя левая ячейка имеет координаты (0, 0).

Точно так же параметр span представляет количество строк и столбцов, которые виджет должен занять в sizer. Это – объект класса wx.GBSpan, но снова, wxPython позволяет вам использовать кортеж (rowspan, colspan). Если span не определен, то значение по умолчанию - (1, 1) - означает, что виджет занимает одну ячейку в каждом направлении.

Например, чтобы поместить виджет во вторую строку и первый столбец, и сделать так, чтобы он занимал три строки и два столбца, вы используете метод Add(widget, (1, 0), (3, 2)) (индексы начинаются с нуля, поэтому вторая строка имеет индекс один, и т.д).

Параметр item метода AddItem – объект класса wx.GBSizerItem, который инкапсулирует всю информацию, необходимую grid bag sizer, чтобы разместить элемент. Маловероятно, что вы будете непосредственно создавать объекты wx.GBSizerItem. Если вы действительно хотите создать такой объект, его конструкторы имеют ту же самую сигнатуру параметров, что и методы Add() grid bag sizer. Как только вы получили wx.GBSizerItem, есть множество методов, которые позволяют вам обращаться к его свойствам. Возможно, самый полезный из них - GetWindow(), который возвращает фактический отображаемый виджет.

Поскольку элементы добавляются к grid bag, используя индексы строки и столбца и количество занимаемых ячеек, порядок, в котором добавлены элементы, может не соответствовать отображению ячеек как в других sizers. Это может стать головной болью – отслеживать какой элемент в какой ячейке фактически отображен. В таблице 11.4 перечислены методы grid bag sizer, облегчающие поиск элементов.

Таблица 11.4 Методы grid bag sizer для управления элементами

Метод	**Описание**
CheckForIntersection(item, excludeItem=None) CheckForIntersection(pos, span, excludeItem=None)	Сравнивает данный элемент или позицию и span со всеми другими элементами в sizer. Возвращает True, если любой из элементов накладывается на позицию элемента или данную позицию и span. Параметр excludeItem - элемент, который исключается при сравнении. Параметр pos - объект wx.GBPosition или кортеж. Параметр span - объект wx. GPSpan или кортеж.
FindItem(window) FindItem(sizer)	Возвращает объект wx.GBSizerItem, соответствующий данному window или sizer. Возвращает None, если window или sizer не находятся в grid bag. Этот метод не будет рекурсивно проверять внутренние sizers.
FindItemAtPoint(pt)	Параметр pt - объект wx.Point или кортеж Python, соответствующий координате точки относительно фрейма. Метод возвращает объект wx.GBSizerItem, содержащий эту точку. Возвращает None, если точка выходит за внешние границы фрейма, или если нет никакого элемента в этой точке.
FindItemAtPosition(pos)	Метод возвращает wx.GBSizerItem в данной позиции ячейки, где pos - объект wx.GBPosition или кортеж Python. Возвращает None, если позиция - вне границ sizer или если нет никакого элемента в этой позиции.
FindItemWithData(userData)	Возвращает wx.GBSizerItem, содержащий объект userData. Возвращает None, если нет такого объекта.

Grid bags также имеют несколько методов, которые могут использоваться для управления размером ячейки и позицией элемента. После отображения grid bag на экране, вы можете использовать метод GetCellSize(row, col), чтобы получить экранный размер данной ячейки. Вы можете получить размер пустой ячейки, используя метод GetEmptyCellSize(), и установить его методом SetEmptyCellSize(sz), где sz - объект wx.Size или кортеж Python.

Вы можете получить позицию или span объекта, который уже находится в grid bag методами GetItemPosition() и GetItemSpan(). Каждый метод принимает window, sizer или index в качестве параметра. Параметр index соответствует индексу в списке sizer после использования методов Add(), который вряд ли будет иметь значение в контексте grid bag. Каждый метод имеет соответствующий Set-метод, SetItemPosition(window, pos) и

SetItemSpan(window, span), в котором первый параметр может быть window, sizer или index, а второй параметр - кортеж Python или объект wx.GBPosition и wx.GBSpan соответственно.

11.3.3 Что такое - box sizer?

Box sizer является самым простым и самым гибким из sizers, предопределенных в wxPython. Box sizer содержит единственный вертикальный столбец или горизонтальную строку, с виджетами, расположенными в линию слева направо или сверху вниз. Он может показаться слишком простым для широкого применения, но его мощь заключается в способности sizers быть вложенными друг в друга. Используя box sizers, вы в состоянии легко поместить различное количество элементов в каждой строке или столбце. И так как каждый sizer - отдельный объект, вы получаете большую гибкость размещения виджетов. Несмотря на кажущуюся простоту, для большинства приложений вертикальный sizer, с вложенными внутри горизонтальными sizers, позволит создавать необходимое вам размещение.

На рисунках 11.11–11.14 показаны несколько примеров простых box sizers. Каждый из этих фреймов был немного изменен пользователем перед снимком экрана, чтобы показать, как каждый sizer реагирует на изменение размеров. На рисунке 11.11 показан горизонтальный box sizer, а на рисунке 11.12 показаны те же самые виджеты в вертикальном box sizer.

Рисунок 11.11

Рисунок 11.12

Рисунок 11.13 показывает вертикальный sizer с одним виджетом, который расширяясь заполняет все доступное пространство. На рисунке 11.14 показан вертикальный sizer с двумя виджетами, заполняющими доступное пространство в различных пропорциях.

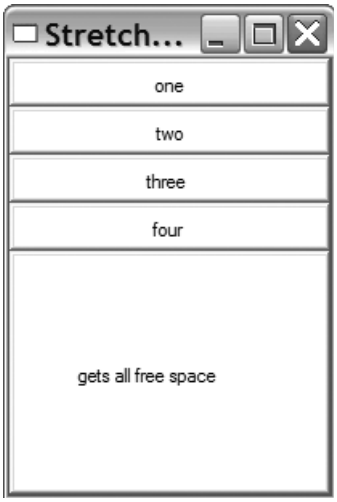

Рисунок 11.13

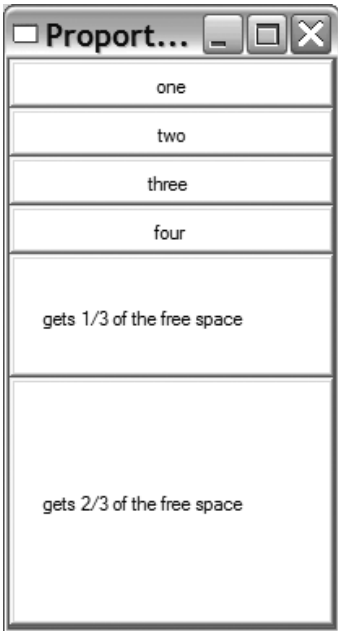

Рисунок 11.14

Код для создания всех четырех фреймов содержится в листинге 11.9.

Листинг 11.9 Создание множества box sizers

```
import wx
from blockwindow import BlockWindow

labels = "one two three four".split()

class TestFrame(wx.Frame):
    title = "none"
    def __init__(self):
        wx.Frame.__init__(self, None, -1, self.title)
        sizer = self.CreateSizerAndWindows()
        self.SetSizer(sizer)
        self.Fit()

class VBoxSizerFrame(TestFrame):
    title = "Vertical BoxSizer"

    def CreateSizerAndWindows(self):
        sizer = wx.BoxSizer(wx.VERTICAL)
        for label in labels:
            bw = BlockWindow(self, label=label, size=(200,30))
            sizer.Add(bw, flag=wx.EXPAND)
        return sizer


class HBoxSizerFrame(TestFrame):
    title = "Horizontal BoxSizer"

    def CreateSizerAndWindows(self):
        sizer = wx.BoxSizer(wx.HORIZONTAL)
        for label in labels:
            bw = BlockWindow(self, label=label, size=(75,30))
            sizer.Add(bw, flag=wx.EXPAND)
        return sizer

class VBoxSizerStretchableFrame(TestFrame):
    title = "Stretchable BoxSizer"

    def CreateSizerAndWindows(self):
        sizer = wx.BoxSizer(wx.VERTICAL)
        for label in labels:
            bw = BlockWindow(self, label=label, size=(200,30))
            sizer.Add(bw, flag=wx.EXPAND)
```

```
        # Add an item that takes all the free space
        bw = BlockWindow(self, label="gets all free space", size=(200,30))
        sizer.Add(bw, 1, flag=wx.EXPAND)
        return sizer

class VBoxSizerMultiProportionalFrame(TestFrame):
    title = "Proportional BoxSizer"

    def CreateSizerAndWindows(self):
        sizer = wx.BoxSizer(wx.VERTICAL)
        for label in labels:
            bw = BlockWindow(self, label=label, size=(200,30))
            sizer.Add(bw, flag=wx.EXPAND)

        # Add an item that takes one share of the free space
        bw = BlockWindow(self,
                label="gets 1/3 of the free space",
                size=(200,30))
        sizer.Add(bw, 1, flag=wx.EXPAND)

        # Add an item that takes 2 shares of the free space
        bw = BlockWindow(self,
                label="gets 2/3 of the free space",
                size=(200,30))
        sizer.Add(bw, 2, flag=wx.EXPAND)
        return sizer

app = wx.PySimpleApp()
frameList = [VBoxSizerFrame, HBoxSizerFrame,
             VBoxSizerStretchableFrame,
             VBoxSizerMultiProportionalFrame]
for klass in frameList:
    frame = klass()
    frame.Show()
app.MainLoop()
```

После просмотра предыдущих примеров sizer, должн быть понятен смысл приведенного выше кода. Box sizers – объекты класса wx.BoxSizer, производного от wx.Sizer. Класс wx.BoxSizer не добавляет почти никаких новых методов. Конструктор для wx.BoxSizer принимает один параметр:

```
wx.BoxSizer(orient)
```

Параметр orient определяет ориентацию sizer, и может принимать значение wx.VERTICAL или wx.HORIZONTAL. Единственный новый метод, определенный для box sizers - GetOrientation(). Он возвращает значение константы, которое было установлено в конструкторе. Вы не можете изменить ориентацию box sizer после создания. А в остальном box sizer используют те же общие методы sizer, которые мы обсуждали ранее в этой главе.

Алгоритм размещения для box sizer обрабатывает первичное направление sizer (определенное его ориентацией при создании) по-другому, в отличие от его вторичного направления. В частности параметр proportion применяется только, когда sizer увеличивается или уменьшается по его первичному направлению, в то время как флаг wx.EXPAND применяется только, когда sizer изменяет размер во вторичном направлении. Другими словами, когда вертикальный box sizer растянут вертикально, параметр пропорции определяет, как каждый элемент будет расти или сжиматься вертикально. Аналогично для горизонтального box sizer. С другой стороны, ростом во вторичном направлении можно управлять, используя флаг wx.EXPAND. Таким образом элементы в вертикальном box sizer будут увеличиваться горизонтально, если им установили флаг wx.EXPAND, иначе элементы остаются в их минимальном или лучшем размере. Рисунок 6.7 в главе 6 иллюстрирует этот процесс.

Пропорциональные изменения в box sizers работают почти так же, как в flex grid sizers, с некоторыми исключениями. Во первых, в box sizer вы определяете пропорциональное поведение, используя параметр proportion при добавлении виджета в sizer — вы не должны отдельно определять его как изменяемый, как в flex grid sizer. Во вторых, поведение sizer для пропорции 0 отлично. В box sizer, пропорция 0 означает, что виджет не будет изменен в первичном измерении, но он все еще может изменяться во вторичном измерении, если используется флаг wx.EXPAND. Когда box sizer вычисляет размещение своих элементов для первичного измерения, он сначала вычисляет место, требуемое неизменяемым элементам с пропорцией 0. Остающееся место делится между пропорциональными элементами. Причем элементы с большим значением пропорции, получают больше места.

11.3.4 Что такое - static box sizer?

Static box sizer – это комбинация box sizer с элементом управления static box. Static box обеспечивает привлекательную рамку вокруг sizer и текстовый заголовок. На рисунке 11.15 показаны три static box sizers в действии.

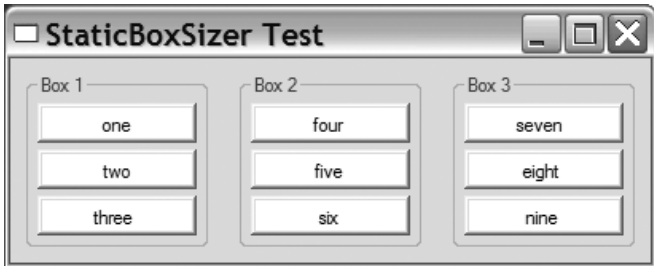

Рисунок 11.15

Листинг 11.10 показывает код для создания static box sizers. Есть две интересных особенности, на которые стоит обратить внимание. Первая - вы должны создать объект static box отдельно от sizer, и вторая - этот пример показывает, как вы могли бы использовать вложенные box sizers. В нашем случае, есть три вертикальных static box sizers, помещенных в горизонтальный box sizer.

Листинг 11.10 Пример static box sizer

```
import wx
from blockwindow import BlockWindow

labels = "one two three four five six seven eight nine".split()

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "StaticBoxSizer Test")
        self.panel = wx.Panel(self)

        # make three static boxes with windows positioned inside them
        box1 = self.MakeStaticBoxSizer("Box 1", labels[0:3])
        box2 = self.MakeStaticBoxSizer("Box 2", labels[3:6])
        box3 = self.MakeStaticBoxSizer("Box 3", labels[6:9])

        # We can also use a sizer to manage the placement of other
        # sizers (and therefore the windows and sub-sizers that they
        # manage as well.)
        sizer = wx.BoxSizer(wx.HORIZONTAL)
        sizer.Add(box1, 0, wx.ALL, 10)
        sizer.Add(box2, 0, wx.ALL, 10)
        sizer.Add(box3, 0, wx.ALL, 10)

        self.panel.SetSizer(sizer)
        sizer.Fit(self)
```

```
    def MakeStaticBoxSizer(self, boxlabel, itemlabels):
        # first the static box
        box = wx.StaticBox(self.panel, -1, boxlabel)

        # then the sizer
        sizer = wx.StaticBoxSizer(box, wx.VERTICAL)

        # then add items to it like normal
        for label in itemlabels:
            bw = BlockWindow(self.panel, label=label)
            sizer.Add(bw, 0, wx.ALL, 2)

        return sizer


app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

Static box sizer – это объект класса wx.StaticBoxSizer, который является производным от wx.BoxSizer. Конструктор принимает два параметра: элемент управления static box и ориентацию:

```
wx.StaticBoxSizer(box, orient)
```

В этом конструкторе параметр orient может принимать такие же значения, как в базовом классе wx.BoxSizer, и параметр box – объект класса wx.StaticBox. Есть только один дополнительный метод, определенный для static box sizers. Это GetStaticBox(), который возвращает объект wx.StaticBox, связанный с sizer. Вы не можете изменить static box после создания sizer.

Класс wx.StaticBox имеет типичный конструктор для элементов управления wxPython. Многие из параметров имеют полезные значения по умолчанию и могут игнорироваться.

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

Для использования в static box sizer, вы не должны устанавливать параметры pos, size, style или name, так как позиция и размер будут управляться sizer, и нет никаких уникальных флажков стиля для wx.StaticBox. Это делает конструктор более простым:

```
box = wx.StaticBox(self.panel, -1, boxlabel)
```

Теперь, когда мы рассмотрели различные виды sizers, мы собираемся показать вам, как можно их использовать в реальном размещении виджетов. См. также главу 6, в которой показан другой пример использования sizers для создания сложного размещения.

11.4 Пример использования sizers в настоящем приложении

Пока, примеры использования sizers, которые мы рассмотрели, были искусственно созданы, чтобы показать функциональные возможности sizers. Однако, у вас мог возникнуть вопрос, как использовать sizers для построения реального размещения, и мы надеемся, что следующий пример подскажет вам некоторые идеи. На рисунке 11.16 показано не очень сложное размещение, построенное с использованием различных sizers.

Рисунок 11.16

Код, используемый для создания рисунка 11.16, показан в листинге 11.11. Просмотрите его для ознакомления. Далее мы рассмотрим его подробно.

Листинг 11.11 Использование sizers для построения формы ввода адреса

```
import wx

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Real World Test")
        panel = wx.Panel(self)

        # (1) First create the controls
        topLbl = wx.StaticText(panel, -1, "Account Information")
        topLbl.SetFont(wx.Font(18, wx.SWISS, wx.NORMAL, wx.BOLD))

        nameLbl = wx.StaticText(panel, -1, "Name:")
        name = wx.TextCtrl(panel, -1, "");

        addrLbl = wx.StaticText(panel, -1, "Address:")
        addr1 = wx.TextCtrl(panel, -1, "");
        addr2 = wx.TextCtrl(panel, -1, "");

        cstLbl = wx.StaticText(panel, -1, "City, State, Zip:")
        city  = wx.TextCtrl(panel, -1, "", size=(150,-1));
        state = wx.TextCtrl(panel, -1, "", size=(50,-1));
        zip   = wx.TextCtrl(panel, -1, "", size=(70,-1));

        phoneLbl = wx.StaticText(panel, -1, "Phone:")
        phone = wx.TextCtrl(panel, -1, "");

        emailLbl = wx.StaticText(panel, -1, "Email:")
        email = wx.TextCtrl(panel, -1, "");

        saveBtn = wx.Button(panel, -1, "Save")
        cancelBtn = wx.Button(panel, -1, "Cancel")

        # Now do the layout.

        # (2) mainSizer is the top-level one that manages everything
        mainSizer = wx.BoxSizer(wx.VERTICAL)
        mainSizer.Add(topLbl, 0, wx.ALL, 5)
        mainSizer.Add(wx.StaticLine(panel), 0,
                wx.EXPAND|wx.TOP|wx.BOTTOM, 5)

        # (3) addrSizer is a grid that holds all of the address info
        addrSizer = wx.FlexGridSizer(cols=2, hgap=5, vgap=5)
        addrSizer.AddGrowableCol(1)
        addrSizer.Add(nameLbl, 0,
                wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
        addrSizer.Add(name, 0, wx.EXPAND)
```

```
        addrSizer.Add(addrLbl, 0,
                wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
        addrSizer.Add(addr1, 0, wx.EXPAND)
        addrSizer.Add((10,10)) # (4) some empty space
        addrSizer.Add(addr2, 0, wx.EXPAND)

        addrSizer.Add(cstLbl, 0,
                wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)

        # (5) the city, state, zip fields are in a sub-sizer
        cstSizer = wx.BoxSizer(wx.HORIZONTAL)
        cstSizer.Add(city, 1)
        cstSizer.Add(state, 0, wx.LEFT|wx.RIGHT, 5)
        cstSizer.Add(zip)
        addrSizer.Add(cstSizer, 0, wx.EXPAND)
        # (6)
        addrSizer.Add(phoneLbl, 0,
                wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
        addrSizer.Add(phone, 0, wx.EXPAND)
        addrSizer.Add(emailLbl, 0,
                wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
        addrSizer.Add(email, 0, wx.EXPAND)

        # (7) now add the addrSizer to the mainSizer
        mainSizer.Add(addrSizer, 0, wx.EXPAND|wx.ALL, 10)

        # (8) The buttons sizer will put them in a row with resizeable
        # gaps between and on either side of the buttons
        btnSizer = wx.BoxSizer(wx.HORIZONTAL)
        btnSizer.Add((20,20), 1)
        btnSizer.Add(saveBtn)
        btnSizer.Add((20,20), 1)
        btnSizer.Add(cancelBtn)
        btnSizer.Add((20,20), 1)

        mainSizer.Add(btnSizer, 0, wx.EXPAND|wx.BOTTOM, 10)

        panel.SetSizer(mainSizer)

        # Fit the frame to the needs of the sizer.  The frame will
        # automatically resize the panel as needed.  Also prevent the
        # frame from getting smaller than this size.
        mainSizer.Fit(self)
        mainSizer.SetSizeHints(self)


app = wx.PySimpleApp()
TestFrame().Show()
app.MainLoop()
```

(1) В первой части кода идет создание виджетов, используемых в окне. Мы создаем все виджеты перед добавлением их в sizers.

(2) Первый sizer в этом размещении - mainSizer, вертикальный box sizer. Первые элементы, добавленные в mainSizer – заголовок (static text label) и разделительная линия (static line).

(3) Следующий sizer - addrSizer, который является flex grid sizer с двумя столбцами. Он используется для размещения адресной информации. Левый столбец addrSizer предназначен для заголовков полей, а правый содержит элементы управления для ввода текста. Это означает, что метки и элементы управления должны быть добавлены именно в таком порядке, чтобы сохранить сетку правильной. Как видите, nameLbl, name, addrLbl и addr1 - первые четыре элемента, добавленные в flex grid.

(4) Следующая строка немного отличается, так как вторая строка адреса не имеет заголовка. В этом случае добавляется пустое пространство размером (10, 10), и затем элемент управления addr2.

(5) Следующая строка тоже отличается, поскольку в строке “City, State, Zip” нужно разместить три текстовых элемента управления. Для этого создается горизонтальный box sizer - cstSizer. Три элемента управления добавляются в cstSizer, и затем этот box sizer добавляется в addrSizer.

(6) В flex grid sizer добавляется телефон и e-mail.

(7) Flex sizer с адресной информацией добавляется в главный sizer.

(8) Кнопки добавляются как горизонтальный box sizer с дополнительными пустыми элементами для разделения. Заметьте, что разделительным элементам устанавливают пропорцию 1, а кнопки оставляют со значением по умолчанию 0.

На этом заканчивается размещение элементов. После этого устанавливается минимальный размер фрейма.

Перед чтением следующего параграфа или выполнением примера, попробуйте выяснить, как фрейм будет реагировать на увеличение размера в горизонтальном и вертикальном направлении.

Если окно будет изменяться вертикально, то ни один из элементов не будет изменяться. Потому что главный sizer – это вертикальный box sizer, и изменение происходит в его первичном направлении, а ни один из его элементов верхнего уровня не был добавлен с пропорцией, большей чем ноль. Если окно изменяется горизонтально, главный sizer изменяется во вторичном направлении, и поэтому все его элементы с флагом wx.EXPAND изменяются горизонтально. Flex grid sizer для адресов определяет столбец с индексом 1 как изменяемый, это означает, что второй столбец, содержащий текстовые элементы управления будет изменяться. В строке “City, State, Zip” элемент City, у которого пропорция 1, будет изменяться, в то время как State и Zip останутся неизменными. Размер кнопок не изменится, так как они имеют пропорцию 0, но пустые разделители перед, между и после кнопок равномерно распределят между собой дополнительное горизонтальное пространство, так как каждый из них имеют пропорцию 1.

Так, если бы вы предположили, что увеличенное окно было бы похоже на рисунок 11.17, то вы бы оказались правы.

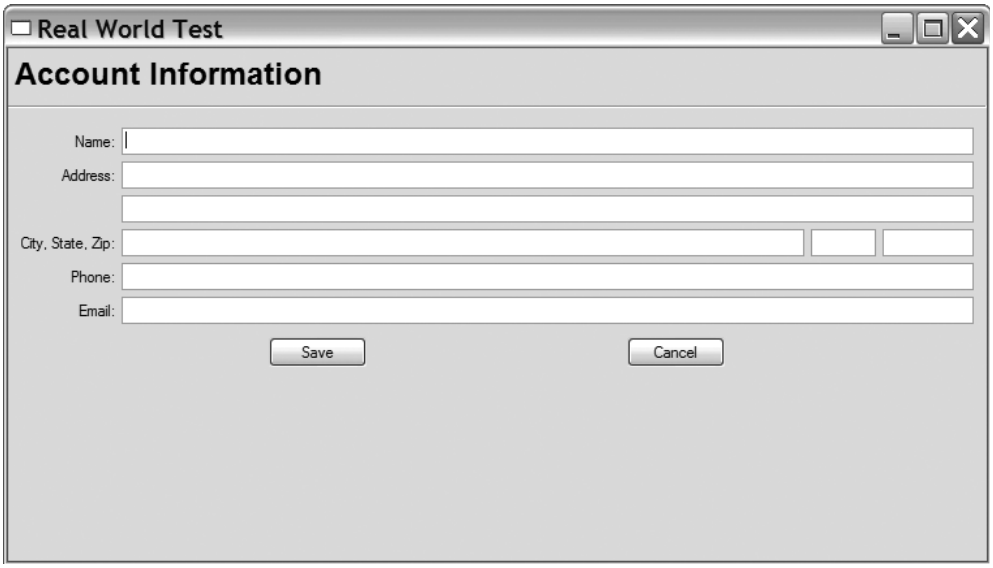

Рисунок 11.17

Заметьте, как элементы, которые мы хотели увеличить горизонтально, фактически увеличились, но виджеты все еще отображаются в тех же самых позициях относительно друг друга. Несмотря на изменение, окно все еще выглядит хорошо и все еще пригодно для использования.

11.5 Резюме

- Sizers - решение проблемы управления размещением виджетов в программе wxPython. Вместо того, чтобы вручную указывать размер и позицию каждого элемента, вы можете добавить элементы в sizer, и sizer будет отвечать за размещение элементов на экране. Sizers особенно хороши в управлении размещением, когда пользователь изменяет размеры фрейма вручную.

- Все sizers wxPython – это объекты классов производных от wx.Sizer. Чтобы использовать sizer, вы должны связать его с контейнерным виджетом. Виджеты, добавленные к контейнеру, вы должны также добавить в sizer. Наконец, вы вызываете метод Fit(), чтобы запустить алгоритм sizer для размещения объектов.

- Все sizers имеют информацию о минимальном размере для каждого из дочерних объектов. Каждый sizer использует различный механизм для размещения виджетов, таким образом одна и та же группа виджетов будет выглядеть по-разному в различных sizers.

- Самый простой sizer в wxPython – это grid sizer (wx. GridSizer). В grid sizer элементы помещаются в двумерную сетку по порядку, в котором они добавляются в sizer. Заполнение сетки начинается с левого верхнего угла, далее по строке вправо и вниз, при заполнении очередной строки. Как правило, вы устанавливаете количество столбцов в сетке, а sizer определяет какое количество строк необходимо, хотя вы можете определить оба измерения, если хотите.

- Все sizers имеют различные методы для добавления виджетов. Так как порядок, в котором виджеты добавляются в sizer, важен при размещении, используются различные методы, чтобы добавить новый виджет в начало, в конец или в середину списка. После того как виджет добавлен в sizer, могут быть установлены другие свойства, которые управляют поведением виджета при изменении sizer. Sizer также может быть сконфигурирован так, чтобы поместить промежуток вокруг некоторых или всех сторон объекта.