wxPython in Action
NOEL RAPPIN, ROBIN DUNN

Перевод: Володеев Сергей

Глава 14. Работа с сеткой (grid control)

Эта глава включает

- Создание сетки
- Добавление строк и ячеек, управление заголовками столбцов
- Управление отображением ячеек
- Создание собственных редакторов
- Обработку пользовательских событий

Сетка (grid) – это, наверное, самый сложный и гибкий виджет в wxPython. В этой главе мы покажем, как работать со многими из особенностей сетки. Мы поговорим о том, как вводить данные и управлять ими в сетке, а также о создании собственных редакторов. Сетка позволяет отображать данные в формате подобном электронной таблице. Этот виджет позволяет вам определять заголовки для строк и столбцов, изменять размер сетки, определять шрифт и цвет для каждой ячейки.

В большинстве случаев, значения в сетке отображаются как простые строки. Однако, вы можете определить свой способ отображения для любой ячейки, который позволит вам показать данные по-другому; например, отображать логическое значение как переключатель. Вы можете редактировать значение ячейки, непосредственно в сетке, и можете использовать другие типы редакторов для различных видов данных. Вы можете также создать ваш собственный редактор и способ отображения, чтобы иметь почти неограниченную гибкость в отображении и управлении данными ячейки. Сетки также имеют большое количество событий мыши и клавиатуры, которые вы можете перехватить и обработать в своем приложении.

Мы начнем наше обсуждение, показав два способа создания сетки.

14.1 Создание вашей сетки

Сетка отображает двумерный набор данных. Для управления и отображения данных, вы должны связать эти данные с сеткой. В wxPython, есть два различных механизма работы с данными в сетке. При этом добавление, удаление и редактирование данных обрабатывается по-разному.

- Данные могут находиться непосредственно в сетке.
- Сетка может управлять данными косвенно, используя таблицу сетки.

Проще когда сетка содержит данные непосредственно. Но в этом случае, сетка должна поддерживать собственную копию данных. Это может быть неудобно, если данных слишком много, или если ваше приложение уже имеет существующую подобную таблице структуру данных. Если так, вы можете использовать таблицу сетки. См. главу 5, в которой дан обзор паттерна MVC.

14.1.1 Как создать простую сетку?

Хотя сетка имеет огромное количество методов, для начала достаточно всего нескольких. На рисунке 14.1 изображена типичная сетка, с некоторыми данными.

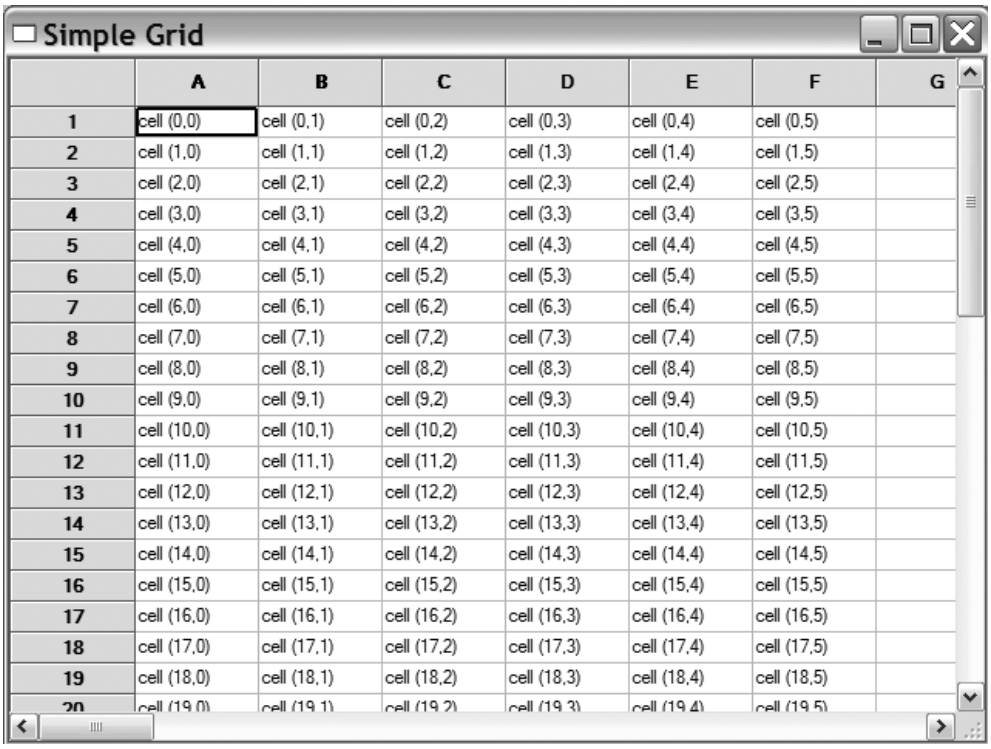

Рисунок 14.1

Сетка – это объект класса wx.grid.Grid. Из-за размера этого класса и связанных классов, а также из-за того, что много программ его не используют, классы для сеток wxPython находятся в их собственном модуле, который автоматически не импортирован. Конструктор для wx.grid.Grid подобен другим конструкторам виджетов.

```
wx.grid.Grid(parent, id, pos=wx.DefaultPosition,
    size=wx.DefaultSize, style=wx.WANTS_CHARS,
    name=wxPanelNameStr)
```

Все эти параметры подобны основному конструктору wx.Window, и имеют то же самое значение. Стиль wx.WANTS_CHARS - значение по умолчанию для сетки; кроме того, wx.grid.Grid не определяет никаких собственных флажков стиля. Так как класс сетки очень сложен, сетка в вашем приложении, скорее всего будет реализована через свой подкласс, вместо того, чтобы использовать объект wx.grid.Grid непосредственно.

В отличие от других виджетов, недостаточно вызвать конструктор, чтобы создать пригодную для использования сетку. Есть два способа инициализировать сетку:

- CreateGrid()
- SetTable()

В этом разделе мы обсудим первый метод, второй метод будет охвачен при обсуждении таблицы сетки.

Чтобы явно инициализировать сетку, используйте метод CreateGrid(numRows, numCols, selmode=wx.grid.Grid.SelectCells). Этот метод нужно вызвать после конструктора, и перед отображением сетки. Первые два параметра, numRows и numCols, определяют начальный размер сетки. Третий параметр, selmode, определяет, как выбирются ячейки. Значение по умолчанию, wx.grid.Grid.SelectCells, означает, что пользователь может выбирать отдельные ячейки. Другие значения - wx.grid.Grid.SelectRows, означает, что выбираются целиком строки, и wx.grid.Grid.SelectionColumns - выбираются столбцы. После создания, вы можете получить режим выбора методом GetSelectionMode(), и установить режим

методом SetSelectionMode(). Вы можете получить количество строк и столбцов, используя методы GetNumberCols() и GetNumberRows().

При инициализации сетки методом CreateGrid(), wxPython создает двумерный массив строк. Как только сетка инициализирована, вы можете поместить в нее данные, используя метод SetCellValue(row, col, s). Параметры row и col - координаты ячейки, и s - текст строки. Если вы хотите получить значение из определенной ячейки, вы можете использовать функцию GetCellValue(row, col), которая возвращает строку. Чтобы очистить всю сетку сразу, используйте метод ClearGrid(). Листинг 14.1 содержит код, создающий сетку, изображенную на рисунке 14.1.

Листинг 14.1 Сетка, созданная с использованием CreateGrid()

```
import wx
import wx.grid

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, title="Simple Grid",
                          size=(640,480))
        grid = wx.grid.Grid(self)
        grid.CreateGrid(50,50)
        for row in range(20):
            for col in range(6):
                grid.SetCellValue(row, col,
                                  "cell (%d,%d)" % (row, col))

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

Использование CreateGrid() и SetCellValue(), для создания и заполнения сетки ограничивается случаем, когда ваши данные составлены из простых строк. Если ваши данные более сложны, или если таблица является особенно большой, вы, вероятно, предпочтете способ, который рассматривается далее.

14.1.2 Как создать сетку с таблицей сетки?

В более сложных случаях, вы можете сохранить ваши данные в таблице сетки, которая является отдельным классом для хранения данных и взаимодействует с сеткой, чтобы отобразить эти данные. Таблица сетки особенно рекомендуется если:

- имеются сложные структуры данных
- данные уже сохранены в других объектах в вашей системе
- таблица является достаточно большой и не умещается вся в памяти

В главе 5, мы обсуждали таблицы сетки в контексте паттерна MVC, наряду с другими способами создания сетки в вашем приложении. В этой главе, мы сосредоточимся на особенностях использования таблицы сетки. На рисунке 14.2 изображена сетка, созданная с использованием таблицы сетки.

Чтобы использовать таблицу сетки, создайте ваш собственный подкласс wx.grid.PyGridTableBase. Ваш подкласс должен переопределить несколько методов родительского класса wx.grid.GridTableBase. Листинг 14.2 содержит код создания сетки, изображенной на рисунке 14.2.

Листинг 14.2 Использование механизма таблицы сетки

```
import wx
import wx.grid
```

```
class TestTable(wx.grid.PyGridTableBase):

    def __init__(self):
        wx.grid.PyGridTableBase.__init__(self)
        self.data = { (1,1) : "Here",
                      (2,2) : "is",
                      (3,3) : "some",
                      (4,4) : "data",
                      }

        self.odd=wx.grid.GridCellAttr()
        self.odd.SetBackgroundColour("sky blue")
        self.odd.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))

        self.even=wx.grid.GridCellAttr()
        self.even.SetBackgroundColour("sea green")
        self.even.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))


    # these five are the required methods
    def GetNumberRows(self):
        return 50

    def GetNumberCols(self):
        return 50

    def IsEmptyCell(self, row, col):
        return self.data.get((row, col)) is not None

    def GetValue(self, row, col):
        value = self.data.get((row, col))
        if value is not None:
            return value
        else:
            return ''

    def SetValue(self, row, col, value):
        self.data[(row,col)] = value


    # the table can also provide the attribute for each cell
    def GetAttr(self, row, col, kind):
        attr = [self.even, self.odd][row % 2]
        attr.IncRef()
        return attr


class TestFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Table",
                          size=(640,480))

        grid = wx.grid.Grid(self)

        table = TestTable()
        grid.SetTable(table, True)


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

В листинге 14.2, вся специфическая для приложения логика была перемещена в класс таблицы сетки. Таким образом, нет никакой потребности создавать собственный подкласс wx.grid.Grid.

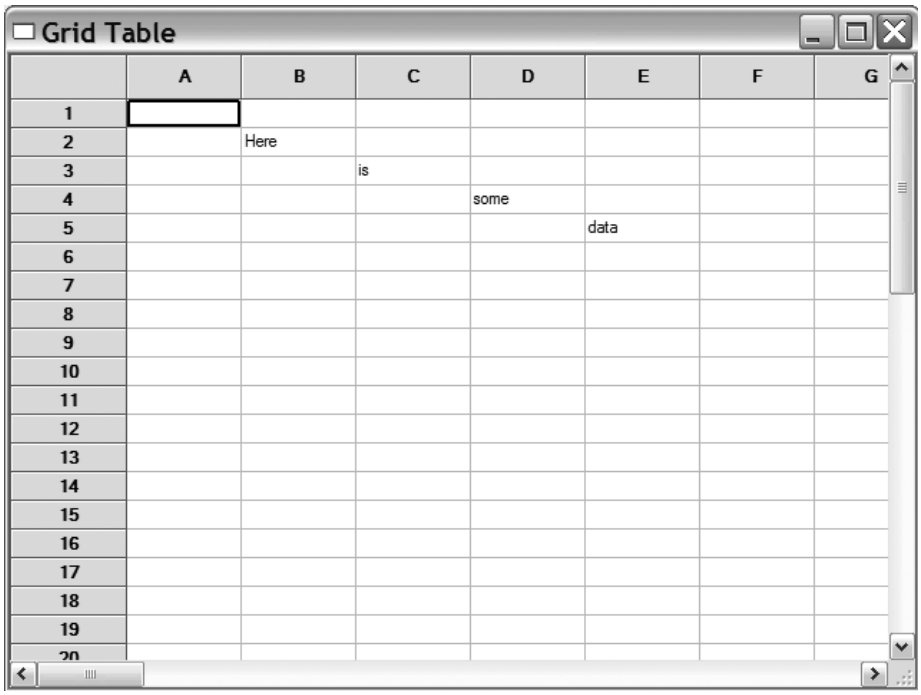

Рисунок 14.2

Для таблицы сетки вы должны переопределить пять методов. Таблица 14.1 содержит список этих методов. Далее мы увидим, что есть другие методы таблицы сетки, которые вы можете переопределить, чтобы дать вашей таблице больше функциональных возможностей.

Таблица 14.1 Методы, переопределяемые в wx.grid.GridTableBase

Метод	**Описание**
GetNumberCols()	Возвращает количество столбцов в сетке
GetNumberRows()	Возвращает количество строк в сетке
GetValue(row, col)	Возвращает значение ячейки с координатами row и col
IsEmptyCell(row, col)	Возвращает True, если ячейка с координатами row и col пустая. В большинстве случаев этот метод возвращает False.
SetValue(row, col, value)	Метод позволяет обновлять вашу основную структуру данных, чтобы она соответствовала представлению. Для таблицы только для чтения, вы также должны определить этот метод, но вы можете сделать так, чтобы он ничего не делал. Этот метод автоматически вызывается, когда пользователь редактирует ячейку.

Для присоединения таблицы сетки к объекту сетки служит метод SetTable(table, takeOwnership=False, selmode=wx.grid.Grid.SelectCells). Используйте его вместо метода CreateGrid(). Параметр table - ваш объект wx.grid.PyGridTableBase. Параметр takeOwnership говорит сетке, что у нее есть связанная таблица. Если этот параметр - True, таблица будет удалена wxPython системой при удалении сетки. Параметр selmode работает также как в CreateGrid().

Есть еще несколько методов, которые вы можете переопределить, чтобы управлять различными частями сетки. Позже в этой главе, мы обсудим некоторые из них наряду с другими функциональными возможностями. И, мы увидим, что в некоторых случаях сетка, созданная с SetTable() ведет себя немного иначе, чем сетка созданная с CreateGrid().

Один дополнительный метод, который вы можете переопределить - Clear(). Он вызывается, когда сетка вызывает ClearGrid(). Можно использовать этот метод, чтобы очистить основной источник данных. Теперь, когда вы присоединили данные к сетке, вы можете производить с этими данными различные операции. В следующем разделе, мы покажем вам, как управлять вашей сеткой.

14.2 Работа с сеткой

Как только сетка создана и инициализирована, с ней можно выполнять различные вещи. Ячейки, строки или столбцы могут быть добавлены и удалены. Вы можете добавить заголовки, изменить размер строки или столбца, и программно изменить видимую или отмеченную часть сетки. В следующих разделах, мы расскажем об этих манипуляциях с сеткой.

14.2.1 Как добавить и удалить строки, столбцы и ячейки?

Даже после того, как сетка была создана, вы все еще можете добавлять новые строки и столбцы. Отметьте, что механизм работает по-другому в зависимости от того, как сетка была создана. Вы можете добавить столбец справа от столбцов сетки, используя метод AppendCols(numCols=1). Чтобы добавить строку снизу сетки, используйте аналогичный метод AppendRows(numRows=1).

Если вы хотите добавить строку или столбец не в конец, а в середину сетки, вы можете использовать метод InsertCols(pos=0, numCols=1) или InsertRows(pos=1, numRows=1). В обоих случаях, параметр pos представляет индекс добавляемого элемента. Если numRows или numCols больше, чем один, дальнейшие элементы добавляются правее этой позиции (для столбцов), или ниже (для строк).

Чтобы удалить строку или столбец, вы можете использовать методы DeleteCols(pos=0, numCols=1) и DeleteRows(pos=0, numRows=1). В этих методах, параметр pos - индекс первой строки или столбца, который будет удален. Дальнейшие элементы будут удалены правее или ниже этого индекса соответственно.

Если сетка была инициализирована методом CreateGrid(), то методы описанные выше будут работать. При этом значением ячеек в новых строках или столбцах будут пустые строки. Однако, если сетка была инициализирована методом SetTable(), то объект таблицы сетки должен разрешить изменения в таблице. Потому что некоторые таблицы, возможно, не могут управлять изменением размеров таблицы (например, сетка отображает таблицу базы данных, которую вы не можете редактировать).

Чтобы разрешить изменение, ваша таблица сетки должна переопределить тот же самый метод изменения. Например, если вы вызываете метод сетки InsertCols(), то таблица сетки также должна определить метод InsertCols(pos=0, numCols=1). Метод таблицы сетки возвращает True, чтобы разрешить изменение или False, чтобы запретить. Например, чтобы создать таблицу, у которой предельное количество строк - 50, напишите следующий метод в таблице сетки

```
def AppendRows(self, numRows=1):
    return (self.GetRowCount() + numRows) <= 50
```

Этот метод возвращает True, чтобы разрешить изменение, пока новое общее количество строк меньше или равно 50.

Содержимое новых строк зависит от метода таблицы сетки GetValue(). Если вы позволяете изменение размера в вашей таблице, вы должны убедиться, что метод GetValue() может обработать новый номер строки и столбца.

Некоторые изменения в сетке не отображаются немедленно, а ожидают, когда сетка будет обновлена. Вы можете вызвать немедленное обновление сетки, используя метод ForceRefresh(). Вообще, если вы делаете программное изменение в вашей сетке, то нужно вставить ForceRefresh(), чтобы гарантировать, что ваше изменение было отображено.

Если вы делаете большое количество изменений в сетке, и не хотите, чтобы изображение сетки мерцало в это время, то нужно выполнить эти изменения как пакет, используя метод BeginBatch(). Вызов этого метода увеличивает на единицу внутренний счетчик для сетки. Вы должны компенсировать BeginBatch() более поздним вызовом EndBatch(). Метод EndBatch() уменьшает на единицу внутренний счетчик. Пока счетчик больше нуля, сетка не будет перерисовывать себя. Вызовы этих методов могут быть вложенными.

14.2.2 Как управлять заголовками строк и столбцов сетки?

В wxPython каждая строка и столбец сетки имеет метку (заголовок). По умолчанию, строкам дают числовые метки, начинающиеся с 1, а столбцам дают алфавитные метки, начинающиеся с A и продолжающиеся к Z, потом AA, AB, и так далее. Если вы создаете электронную таблицу – здорово. Но для большинства других приложений это не подходит. К счастью в wxPython есть методы, позволяющие изменить метки. На рисунке 14.3 показана сетка с измененными заголовками.

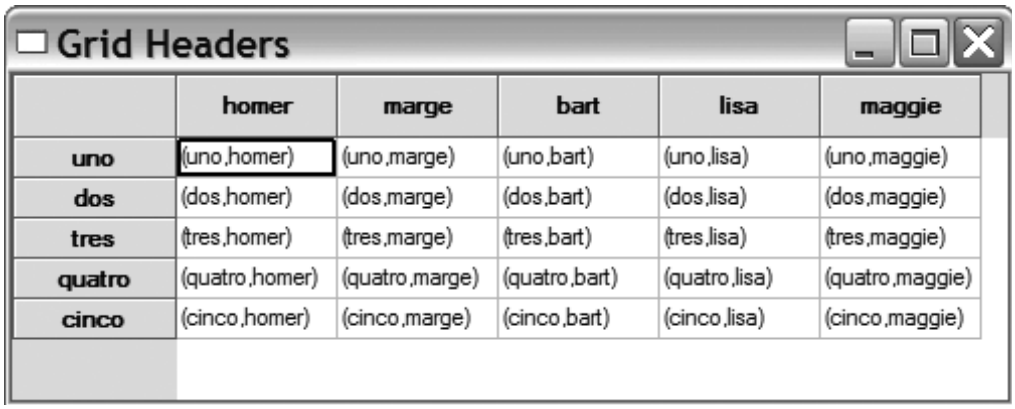

Рисунок 14.3

Листинг 14.3 содержит код для этого рисунка. В этом примере, сетка была инициализирована методом CreateGrid().

Листинг 14.3 Код для простой сетки со своими метками

```
import wx
import wx.grid

class TestFrame(wx.Frame):

    rowLabels = ["uno", "dos", "tres", "quatro", "cinco"]
    colLabels = ["homer", "marge", "bart", "lisa", "maggie"]

    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Headers",
                          size=(500,200))
        grid = wx.grid.Grid(self)
        grid.CreateGrid(5,5)
        for row in range(5):
            grid.SetRowLabelValue(row, self.rowLabels[row])
            grid.SetColLabelValue(row, self.colLabels[row])
            for col in range(5):
                grid.SetCellValue(row, col,
                        "(%s,%s)" % (self.rowLabels[row],
                        self.colLabels[col]))

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

Как добавление и удаление строк, изменение заголовков, имеют отличия для разных типов сеток. Для сеток, которые созданы методом CreateGrid(), установите значения метки, используя методы SetColLabelValue(col, value) и SetRowLabelValue(row, value). параметр col и row - индекс соответствующего столбца или строки, а параметр value - строка, которая будет отображена в заголовке. Чтобы получить метки для строки или столбца, используйте методы GetColLabelValue(col) и GetRowLabelValue(row).

Для сетки, использующей внешнюю таблицу сетки, вы можете достигнуть того же самого эффекта, переопределив методы таблицы сетки GetColLabelValue(col) и GetRowLabelValue(row). Эти методы вызываются сеткой, когда необходимо отобразить заголовки, и сетка имеет связанную таблицу. Так как возвращаемое значение определяется вашим методом, нет необходимости переопределять или вызывать соответствующие Set-методы. Хотя эти методы SetColLabelValue(col, value) и SetRowLabelValue(row, value) существуют, они редко используются. Как правило, вы не нуждаетесь в этих методах. В листинге 14.4 показано, как изменять метки в сетке таблицы.

Листинг 14.4 Сетка с таблицей сетки, которая имеет свои заголовки

```
import wx
import wx.grid

class TestTable(wx.grid.PyGridTableBase):
    def __init__(self):
        wx.grid.PyGridTableBase.__init__(self)
        self.rowLabels = ["uno", "dos", "tres", "quatro", "cinco"]
        self.colLabels = ["homer", "marge", "bart", "lisa", "maggie"]

    def GetNumberRows(self):
        return 5

    def GetNumberCols(self):
        return 5

    def IsEmptyCell(self, row, col):
        return False

    def GetValue(self, row, col):
        return "(%s,%s)" % (self.rowLabels[row], self.colLabels[col])

    def SetValue(self, row, col, value):
        pass

    def GetColLabelValue(self, col):
        return self.colLabels[col]

    def GetRowLabelValue(self, row):
        return self.rowLabels[row]

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Table",
                          size=(500,200))
        grid = wx.grid.Grid(self)
        table = TestTable()
        grid.SetTable(table, True)

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

По умолчанию, метки отцентрированы. Однако, вы можете изменить такое поведение, используя методы SetColumnLabelAlignment(horiz, vert) и SetRowLabelAlignment(horiz,

vert). Параметр horiz управляет горизонтальным выравниванием и может иметь значения wx.ALIGN_LEFT, wx.ALIGN_CENTRE или wx.ALIGN_RIGHT. Параметр vert управляет вертикальным выравниванием и может иметь значения wx.ALIGN_TOP, wx.ALIGN_CENTRE или wx.ALIGN_BOTTOM.

Заголовки строк и столбцов совместно используют общие свойства шрифта и цвета. Вы можете управлять этими свойствами с помощью методов SetLabelBackgroundColour(colour), SetLabelFont(font) и SetLabelTextColour(colour). Параметр colour - это объект wx.Colour или строка с названием цвета, которую wxPython преобразует в цвет. Параметр font - объект wx.Font. Существуют соответствующие методы для получения этих атрибутов GetLabelBackgoundColour(), GetLabelFont() и GetLabelTextFont().

14.2.3 Как управлять размером элементов сетки?

У сетки есть несколько различных методов для изменения размеров ячеек, строк и столбцов. В этом разделе мы обсудим каждый из этих методов. На рисунке 14.4 изображены некоторые из способов изменения размеров ячейки.

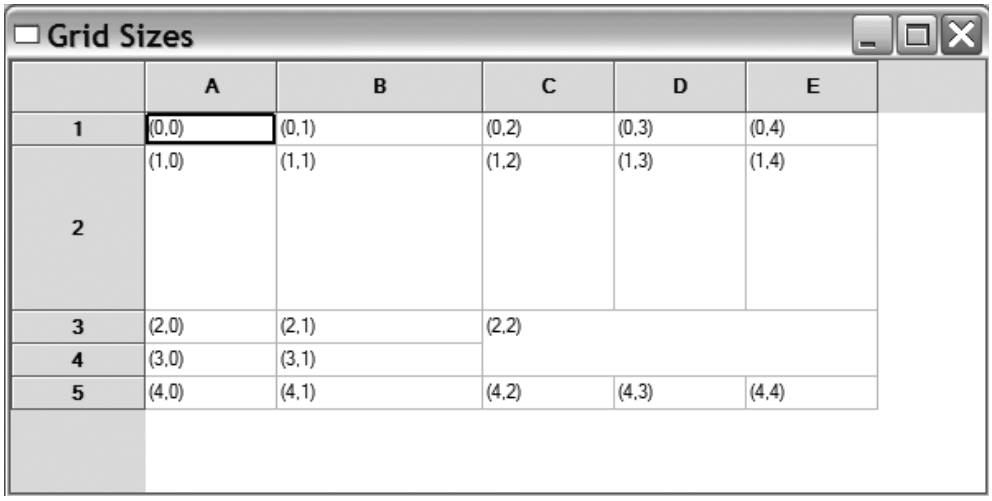

Рисунок 14.4

В листинге 14.5 создается сетка с измененными ячейками, строками и столбцами.

Листинг 14.5 Изменение размеров ячеек, строк и столбцов

```
import wx
import wx.grid

class TestFrame(wx.Frame):


    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Sizes",
                          size=(600,300))
        grid = wx.grid.Grid(self)
        grid.CreateGrid(5,5)
        for row in range(5):
            for col in range(5):
                grid.SetCellValue(row, col, "(%s,%s)" % (row, col))

        grid.SetCellSize(2, 2, 2, 3)
        grid.SetColSize(1, 125)
        grid.SetRowSize(1, 100)

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

```
app.MainLoop()
```

Изменение размера ячейки

Один из способов изменить размер ячейки состоит в том, чтобы заставить ячейку охватить больше чем одну строку или столбец, аналогично атрибутам HTML rowspan и colspan. Чтобы управлять этим в wxPython, используйте метод SetCellSize(row, col, num_rows, num_cols). Он заставляет ячейку охватывать num_rows строк и num_col столбцов. При нормальных обстоятельствах, каждая ячейка занимает одну строку и один столбец, поэтому, чтобы заставить ячейку заполнить большее пространство, вы должны передать значения параметров большее единицы. Задание нулевых или отрицательных значений для num_rows и num_cols, вызовет ошибку. Если вы устанавливаете размер ячейки, который накладывается на другую измененную ячейку, то, предварительно, нужно отменить изменения другой ячейки. Вы можете также отключить отображение переполнения ячейки, используя метод SetCellOverflow(row, col, allow). Вызов этого метода препятствует ячейке переполняться, даже если ее размер был установлен методом SetCellSize().

Более типичный метод изменения размеров – это установка размеров в пикселях строки или столбца. Вы можете установить размер определенной строки или столбца, используя методы SetColSize(col, width) и SetRowSize(row, height). Естественно, вы можете определить текущий размер строки или столбца, используя GetColSize(col) или GetRowSize(row).

Установка размеров по умолчанию

Вы можете изменить размер всей сетки, изменяя размер по умолчанию для всех строк или всех столбцов, используя методы:

```
SetDefaultColSize(width, resizeExistingCols=False)
SetDefaultRowSize(height, resizeExistingRows=False)
```

В обоих методах, первый параметр - новый размер в пикселях. Если второй параметр - True, то размеры всех существующих строк или столбцов будут немедленно изменены в соответствии с новым значением по умолчанию. Если параметр - False, существующие строки или столбцы не изменяются, и новое значение по умолчанию применяется только к новым строкам или столбцам. Обычно размеры по умолчанию устанавливаются в начале инициализации, даже перед вызовом CreateGrid() или SetTable(). Вы можете получить текущие размеры по умолчанию, используя методы GetDefaultColSize() и GetDefaultRowSize().

Есть отличия в работе установки размеров по умолчанию и установки размеров для индивидуальных строк или столбцов. Для размеров по умолчанию, wxPython должен хранить только два целочисленных значения. Как только вы устанавливаете свой размер для строки или столбца не равный размеру по умолчанию, wxPython переключается и хранит размер каждой строки или столбца в массиве. Если ваша таблица является очень большой, то для этого понадобится существенное количество дополнительной памяти. Примите это к сведению.

Иногда нужно установить минимальный размер для строки или столбца так, чтобы независимо от программных методов или действий пользователя, перетаскивающего линии сетки, строка или столбец не могли стать меньшими определенной величины. Это нужно для того, чтобы текст или число в ячейке не были обрезаны.

В wxPython, вы можете установить минимальное значение для всей сетки или установить его для отдельных строк и столбцов. Чтобы изменить минимальное значение для всей сетки, используйте метод SetColMinimalAcceptableWidth(width) или

SetRowMinimalAcceptableHeight(height). Параметр - минимальный размер для всех строк или столбцов в пикселях. Чтобы установить минимум для отдельного столбца или строки, используйте SetColMinimalWidth(col, width) или SetRowMinimalHeight(row, height). Для этих методов, первый параметр - индекс устанавливаемого по размеру элемента, а второй параметр - новый размер в пикселях. Минимальный размер отдельной строки должен быть больше, чем минимальное значение, установленное для строк всей сетки. Существующие строки или столбцы, которые имеют размер меньший, чем новый минимум, не будут автоматически изменены. Поэтому, вы вероятно захотите вызвать эти методы в процессе инициализации прежде, чем добавить данные. Лучше сохранять минимальные приемлемые значения близкими к фактическим наименьшим размерам реальной ячейки. Каждый из этих методов имеет соответствующую функцию для получения значения.

```
GetColMinimalAcceptableWidth()
GetRowMinimalAcceptableHeight()
GetColMinimalWidth(col)
GetRowMinimalHeight(row)
```

Установка размеров заголовков

Область заголовков сетки имеет отдельный набор функций для установки размеров. В этом случае, вы устанавливаете ширину заголовка строки, и высоту заголовка столбца. При этом метка столбца обрабатывается как специальная строка, а метка строки как специальный столбец. Как пример, посмотрите на электронную таблицу. Метод SetRowLabelSize(width) устанавливает ширину меток строки, а метод SetColLabelSize(height) устанавливает высоту меток столбца. Вы можете получить эти размеры методами GetRowLabelSize() и GetColLabelSize().

Хотелось бы не заботиться о фактическом размере ячеек в пикселях. Желательно, чтобы размеры были установлены автоматически достаточно большими, чтобы отобразить данные. В wxPython вы можете установить авто размер для всей сетки методом AutoSize(). Когда он вызывается, все строки и столбцы изменяются так, чтобы минимально соответствовать их содержанию. Вы можете установить авто размер и для определенных строк и столбцов. Метод AutoSizeColumn(col, setAsMin=True) изменяет размер столбеца, чтобы минимально соответствовать его содержанию. Если параметр setAsMin - True, новый авто размер будет минимальным размером этого столбца. Точно так же метод AutoSizeColumns(setAsMin=True) изменяет размеры всех столбцов в сетке. Есть аналогичные методы и для строк, AutoSizeRow(row, setAsMin=True) и AutoSizeRows (setAsMin=True). Снова, параметр setAsMin позволяет одновременно установить минимальный размер строки.

Вы можете также позволить пользователю изменять размеры строк и столбцов, перетаскивая границы ячейки. Вот набор методов для этого:

- EnableDragColSize(enable=True), позволяет изменять ширину столбцов
- EnableDragRowSize(enable=True), позволяет изменять высоту строк
- EnableDragGridSize(enable=True) управляет обоими измерениями сразу

Используя методы без параметра, мы разрешаем указанные действия. Существуют параллельные методы, которые позволяют отключить эти возможности:

- DisableDragColSize()
- DisableDragRowSize()
- DisableDragGridSize()

И ряд методов для получения состояния:

- CanDragColSize()
- CanDragRowSize()
- CanDragGridSize()

14.2.4 Как можно узнать, какие ячейки выбраны или видимы?

В сетке, одна или более ячеек могут быть выбраны пользователем. В wxPython, есть несколько методов, которые позволяют управлять группой выбранных ячеек. Вы можете также определить позицию ячейки, содержащей курсор, и точную позицию на экране любой ячейки в сетке.

Одновременно, в сетке могут быть выбраны следующие комбинации ячеек:

- коллекция отдельных ячеек
- коллекция строк
- коллекция столбцов
- коллекция блоков ячеек

Пользователь может выбрать больше чем одну группу ячеек, используя клавишу Ctrl или щелчком мыши по ячейкам, заголовку строки или столбца. Чтобы определять, есть ли выбранные ячейки, используйте метод IsSelection(). Вы можете определить, выбрана ли какая-нибудь определенная ячейка, используя метод IsInSelection(row, col).

В таблице 14.2 показаны несколько методов, которые позволяют вам по-разному получить выбранные ячейки.

Таблица 14.2 Методы получения, выбранных ячеек

Метод	**Возвращаемое значение**
GetSelectedCells()	Возвращает список всех ячеек, которые были индивидуально выбраны. Каждый элемент списка - кортеж (row, col).
GetSelectedCols()	Возвращает список индексов столбцов, которые были выбраны, щелчком на заголовке столбца.
GetSelectedRows()	Возвращает список индексов строк, которые были выбраны, щелчком на заголовке строки
GetSelectionBlockTopLeft()	Возвращает список, каждый элемент которого - кортеж (row, col) левого верхнего угла каждого выбранного блока ячеек.
GetSelectionBlockBottomRight()	Возвращает список, каждый элемент которого - кортеж (row, col) правого нижнего угла каждого выбранного блока ячеек.

Есть также несколько методов для того, чтобы установить или изменить выбор. Первый - ClearSelection(), который удаляет любой текущий выбор. После вызова этого метода, IsSelection() возвращает False. Вы можете выполнить противоположное действие выбрав все ячейки методом SelectAll(). Весь столбец или всю строку можно выбрать методами SelectCol(col, addToSelected=False) и SelectRow(row, addToSelected=False). В обоих методах, первый параметр - индекс строки или столбца для выбора. Если параметр addToSelected - True, все другие в данный момент выбранные ячейки остаются выбранными и строка или столбец добавляются к существующему выбору. Если параметр addToSelected – False, то выбор всех других ячеек отменяется. Аналогично можно добавить прямоугольный блок методом SelectBlock(topRow, leftCol, bottomRow, rightCol, addToSelected=False), где первые четыре параметра - углы прямоугольника, и параметр addToSelected ведет себя как в предыдущих методах.

Вы можете определить, видима ли ячейка в текущем отображении, используя метод IsVisible(row, col, wholeCellVisible=True). Метод возвращает True, если ячейка с указанными координатами в настоящий момент отображается на экране, а не скрыта в прокручиваемой области. Если wholeCellVisible - True, вся ячейка должна быть видимой, чтобы метод возвратил True, если параметр - False, достаточно, чтобы была видима любая часть ячейки. Наоборот, метод MakeCellVisible(row, col) гарантирует, что ячейка в данных координатах видима без прокрутки.

В дополнение к выбранным ячейкам, сетка также имеет текущую ячейку (курсор). Вы можете определить текущую позицию курсора методами GetGridCursorCol() и GetGridCursorRow(), которые возвращают соответствующий целочисленный индекс. Вы можете установить курсор явно методом SetGridCursor(row, col). Этот метод перемещает курсор и неявно вызывает MakeCellVisible для новой позиции курсора.

В таблице 14.3 перечислены методы сетки, которые помогают выполнять преобразования между координатами сетки и экранными координатами.

Таблица 14.3 Методы преобразования координат.

Метод	**Описание**
BlockToDeviceRect(topLeft, bottomRight)	Параметры topLeft и bottomRight - координаты ячейки - в wxPython, передаются как кортежи (row, col). Возвращаемое значение - wx.Rect координаты в пикселях прямоугольника, ограниченного данными координатами сетки. В случае необходимости, прямоугольник отсечен по размеру окна сетки.
CellToRect(row, col)	Возвращает wx.Rect с координатами относительно контейнера для ячейки (row, col).
XToCol(x)	Возвращает целочисленный индекс столбца, содержащего данную координату x относительно контейнера. Если нет никакого столбца в этой координате, то возвращает wx.NOT_FOUND.
XToEdgeOfCol(x)	Возвращает целочисленный индекс столбца, правый край которого является самым близким к данной координате x. Если нет такого столбца, возвращает wx.NOT_FOUND.
YToRow(y)	Возвращает целочисленный индекс строки, содержащей данную координату y. Если нет такой строки, возвращает wx.NOT_FOUND.
YToEdgeOfRow(y)	Возвращает строку, нижний край которой является самым близким к данной координате y. Если нет такой строки, возвращает wx.NOT_FOUND.

Вы можете использовать эти методы, чтобы преобразовать позиции щелчка мыши в координаты ячейки сетки, в которой произошел щелчок.

14.2.5 Как изменить цвет или шрифт ячейки сетки?

Как и в других элементах управления, есть ряд свойств, которые вы можете использовать, чтобы изменить атрибуты изображения для каждой ячейки. На рисунке 14.5 изображена часть из того, что можно сделать, используя эти методы.

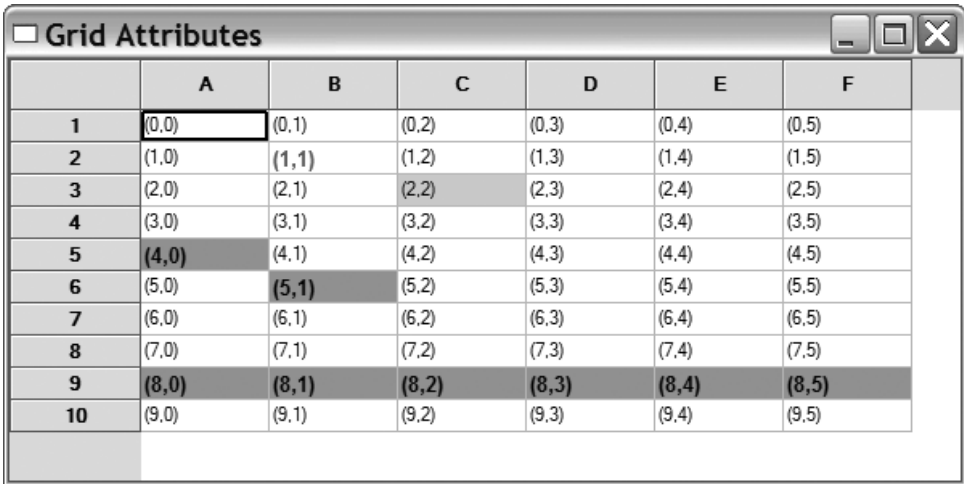

Рисунок 14.5

В листинге 14.6 показан код для рисунка 14.5. Обратите внимание на использование методов сетки, нацеленных на определенную ячейку и создание объектов wx.grid.GridCellAttr.

Листинг 14.6 Изменение цвета ячеек сетки

```
import wx
import wx.grid

class TestFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Attributes",
                          size=(600,300))
        grid = wx.grid.Grid(self)
        grid.CreateGrid(10,6)
        for row in range(10):
            for col in range(6):
                grid.SetCellValue(row, col, "(%s,%s)" % (row, col))

        grid.SetCellTextColour(1, 1, "red")
        grid.SetCellFont(1,1, wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))
        grid.SetCellBackgroundColour(2, 2, "light blue")

        attr = wx.grid.GridCellAttr()
        attr.SetTextColour("navyblue")
        attr.SetBackgroundColour("pink")
        attr.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))

        grid.SetAttr(4, 0, attr)
        grid.SetAttr(5, 1, attr)
        grid.SetRowAttr(8, attr)


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

Мы начнем обсуждение с методов, используемых для установки значений по умолчанию для всей сетки. Вы можете установить выравнивание по умолчанию для всех ячеек в сетке методом SetDefaultCellAlignment(horiz, vert), где horiz - wx.LEFT, wx.CENTRE или wx.RIGHT, и vert - wx.TOP, wx.CENTRE или wx.BOTTOM. Получить, установленное по умолчанию выравнивание, можно с помощью метода GetDefaultCellAlignment(), который возвращает кортеж (horiz, vert).

Цвет фона и текста может быть установлен методами SetDefaultCellTextColour(colour) и SetDefaultCellBackgroundColour(colour). Как обычно, цвет может быть объектом wx.Colour или строкой названием цвета. Методы для получения значений - GetDefaultCellTextColour() и GetDefaultCellBackgroundColour(). Наконец, вы можете управлять шрифтом по умолчанию с помощью методов SetDefaultCellFont(font) и GetDefaultCellFont().

Используя следующие методы, вы можете установить эти же признаки для отдельной ячейки.

```
GetCellAlignment(row, col)
SetCellAlignment(row, col, horiz, vert)

GetCellBackgroundColour(row, col)
SetCellBackgroundColour(row, col, colour)

GetCellFont(row, col)
SetCellFont(row, col, font)

GetCellTextColour(row, col)
SetCellTextColour(row, col, colour)
```

Каждый метод аналогичен методу для значения по умолчанию, с добавлением параметров row и col, которые определяют координаты ячейки.

Выбранные ячейки имеют другой цвет фона и текста, которые могут быть изменены. Методы - SetSelectionBackground(colour) и SetSelectionForeground(colour), и связанные методы GetSelectionBackground() и GetSelectionForeground().

Вы можете поместить дополнительное пространство вокруг сетки при использовании метода SetMargins(extraWidth, extraHeight) - параметры указывают количество пикселей, которые используются, чтобы добавить пространство в пределах ее контейнера.

Внутренне, класс wx.grid.Grid использует класс wx.grid.GridCellAttr, чтобы управлять атрибутами каждой ячейки. Этот класс имеет методы для всех свойств, обсуждаемых в этом разделе. Вы можете получить обект attr отдельной ячейки при использовании метода GetOrCreateCellAttr(row, col) метода, который возвращает объект атрибутов ячейки, или создает такой объект в случае необходимости. Объект атрибутов ячейки создается только, если ячейка определила свойства, отличные от значения по умолчанию для сетки. Как только вы получаете объект атрибутов ячейки, вы можете использовать его, чтобы изменить свойства ячейки.

Чтобы создать ваш собственный объект атрибутов ячейки, используйте конструктор wx.grid.GridCellAttr(). Вы можете установить некоторые параметры, затем передать объект в методы SetColAttr(attr) или SetRowAttr(attr), которые применяют атрибуты к каждой ячейке, как это показано в листинге 14.6.

Если вы используете таблицу сетки, вы можете переопределить метод GetAttr(row, col), чтобы он возвращал объект wx.grid.GridCellAttr для определенной ячейки.

Вы можете также изменить цвет и отображение линий координатной сетки. Отображением линий координатной сетки управляет метод EnableGridLines(enable). Если параметр - True, линии отображаются, если False они не отображаются. Вы можете изменить цвет линий координатной сетки методом SetGridLineColor(colour).

14.3 Управление отображением ячеек и создание собственных редакторов

Что делает сетку настолько гибким и полезным инструментом – это идея, что механизм для отображения или редактирования ячейки может быть изменен на основании значения ячейки. В следующих разделах, мы покажем, как использовать предопределенные способы отображения и редакторы, и как написать ваши собственные.

14.3.1 Как использовать способ отображения (renderer) ячейки?

По умолчанию, сетка отображает ее данные как простые строки, однако, вы можете отображать ваши данные в других форматах. Вы можете захотеть, отображать логические данные как переключатели, или числовые значения в графическом формате.

В wxPython, каждая ячейка может иметь собственный способ отображения (renderer), который позволяет отображать ее данные по-другому. Следующие разделы обсуждают несколько предопределенных способов отображения. А также как определить ваш собственный способ отображения.

Предопределенные способы отображения

Renderer сетки – это объект класса wx.grid.GridCellRenderer, который является абстрактным родительским классом. Как правило, вы используете один из его подклассов. Таблица 14.4 описывает несколько предопределенных renderers, которые вы можете использовать в ваших ячейках. Каждый из этих классов имеет конструктор и соответствующие Get- и Set- методы.

Таблица 14.4 Предопределенные способы отображения (renderers) ячеек сетки

Renderer class	**Описание**
wx.grid.GridCellAutoWrapStringRenderer	Печатает текстовые данные с переходом на новую строку в границах ячейки.
wx.grid.GridCellBoolRenderer	Используется для булевых данных. Отображается в виде переключателя, включенного для значения True, и выключенного для False.
wx.grid.GridCellDateTimeRenderer	Позволяет отображать в ячейке отформатированную дату и/или время.
wx.grid.GridCellEnumRenderer	Отображает номер как текстовый эквивалент. Другими словами, в ячейке находятся данные из списка [0,1,2], но ячейка отображает их как значение из списка ["Джон", "Фред", "Боб"].
wx.grid.GridCellFloatRenderer	Отображает числовые данные с плавающей запятой с определенной шириной и точностью. Конструктор для этого класса берет два параметра (width=-1, precision=-1), где width - минимальное число цифр, для показа, и precision - максимальное число цифр, после десятичной точки. Числа, отображенные этим renderer выровнены по правому краю по умолчанию.
wx.grid.GridCellNumberRenderer	Отдает числовые данные. Числа, отображенные этим renderer выровнены по правому краю по умолчанию.
wx.grid.GridCellStringRenderer	Отображает данные как простые строки. По умолчанию, данные, отображенные этим renderer выровнены по левому краю. Это renderer по умолчанию используемый сеткой для

	всех ячеек.

Чтобы получить renderer для определенной ячейки, используйте метод GetCellRenderer(row, col), который возвращает объект renderer для ячейки с указанными координатами. Чтобы установить renderer для ячейки, используйте SetCellRenderer(row, col, renderer), где параметр renderer - новый renderer для этой ячейки. Эти методы просто устанавливают renderer в объекте атрибутов ячейки, таким образом вы можете работать с GridCellAttr непосредственно, если предпочитаете. Вы можете получить и установить renderer по умолчанию для всей сетки при использовании методов GetDefaultRenderer() и SetDefaultRenderer(renderer).

Вы можете установить renderer для всего столбца в случае, когда определенные столбцы всегда отображают данные одного типа. Методы для этого - SetColFormatBool(col), SetColFormatNumber(col) и SetColFormatFloat(col, width, precision).

Создание своего renderer

Чтобы создавать ваш собственный renderer, нужно создать класс производный от wx.grid.PyGridCellRenderer. Создание собственного renderer позволит вам делать такие вещи как отображение числового значение в виде мини-диаграммы или выполнить специальное форматирование для строки или даты.

На рисунке 14.6 изображен простой собственный renderer, который случайным образом изменяет цвет фона ячейки. Этот renderer установлен для определенной строки. Вы обратите внимание, когда сетка будет мерцать при перерисовке, поскольку выбирается случайный цвет.

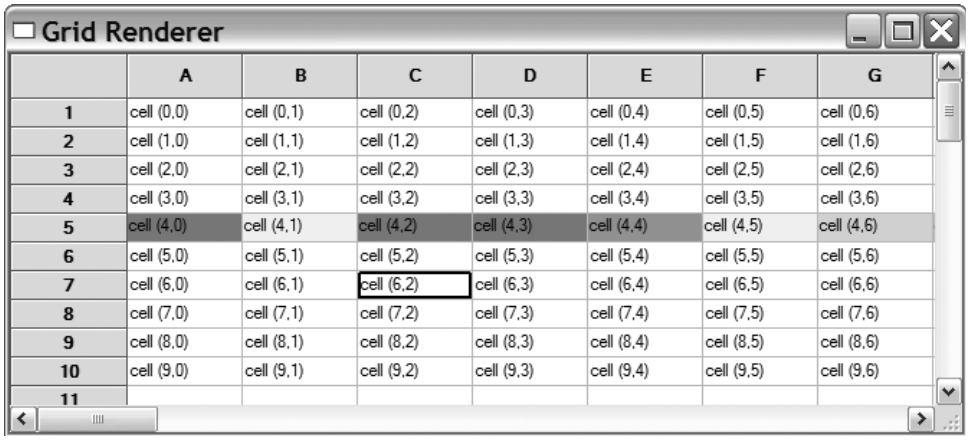

Рисунок 14.6

В листинге 14.7 показано создание собственного класса renderer, и переопределение методов.

Листинг 14.7 Собственный renderer, который изменяет цвет случайным образом

```
import wx
import wx.grid
import random

class RandomBackgroundRenderer(wx.grid.PyGridCellRenderer):
    def __init__(self):
        wx.grid.PyGridCellRenderer.__init__(self)


    def Draw(self, grid, attr, dc, rect, row, col, isSelected):
        text = grid.GetCellValue(row, col)
        hAlign, vAlign = attr.GetAlignment()
```

```
        dc.SetFont( attr.GetFont() )
        if isSelected:
            bg = grid.GetSelectionBackground()
            fg = grid.GetSelectionForeground()
        else:
            bg = random.choice(["pink", "sky blue", "cyan", "yellow",
                                "plum"])
            fg = attr.GetTextColour()

        dc.SetTextBackground(bg)
        dc.SetTextForeground(fg)
        dc.SetBrush(wx.Brush(bg, wx.SOLID))
        dc.SetPen(wx.TRANSPARENT_PEN)
        dc.DrawRectangleRect(rect)
        grid.DrawTextRectangle(dc, text, rect, hAlign, vAlign)


    def GetBestSize(self, grid, attr, dc, row, col):
        text = grid.GetCellValue(row, col)
        dc.SetFont(attr.GetFont())
        w, h = dc.GetTextExtent(text)
        return wx.Size(w, h)

    def Clone(self):
        return RandomBackgroundRenderer()

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Renderer",
                          size=(640,480))

        grid = wx.grid.Grid(self)
        grid.CreateGrid(50,50)

        # Set this custom renderer just for row 4
        attr = wx.grid.GridCellAttr()
        attr.SetRenderer(RandomBackgroundRenderer())
        grid.SetRowAttr(4, attr)

        for row in range(10):
            for col in range(10):
                grid.SetCellValue(row, col,
                                  "cell (%d,%d)" % (row, col))

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

Ваш класс renderer должен переопределить три метода базового класса

- Draw()
- GetBestSize()
- Clone()

Самый важный из этих методов - это метод Draw(grid, attr, dc, rect, row, col, isSelected). Через параметры этого метода передается информация о сетке, в которой нужно нарисовать ячейку. Параметр grid - объект сетки, содержащий эту ячейку. Параметр attr содержит объект атрибутов ячейки. Параметр dc - контекст устройства, чтобы рисовать вы должны использовать его методы. Параметр rect - ограничивающий прямоугольник ячейки в логических координатах. Параметры row и col - координаты ячейки. Параметр isSelected - True, если ячейка в настоящий момент выбрана. В пределах этого метода вы можете делать что хотите. Вы можете вызвать метод базового класса, который рисует фон и устанавливает шрифт и цвет текста.

Второй метод - GetBestSize(grid, attr, dc, row, col). Этот метод возвращает объект wx.Size, который представляет предпочтительный размер ячейки для тех данных, размер которых вы можете вычислить самостоятельно. Параметры grid, attr, dc, row и col аналогичны параметрам метода Draw().

Наконец, вы должны определить метод Clone(), который возвращает объект wx.grid.GridCellRenderer, который должен быть равным вызываемому объекту. Как только renderer определен, Вы можете использовать его точно так же как предопределенный renderer, устанавливая его как renderer для определенных ячеек.

14.3.2 Как редактировать ячейку?

В wxPython сетка позволяет редактировать значения непосредственно в ячейке. Это поведение по умолчанию. Щелкните мышью на ячейке, или начните печатать новое значение, и откроется редактор, который позволит ввести другую строку. В этом разделе, мы обсудим множество способов изменить это поведение по умолчанию.

Вы можете отключить редактирование для всей сетки методом EnableEditing(enable). Если параметр enable - False, то никакая ячейка в сетке не может быть доступной для редактирования. Если редактирование выключено этим методом, то оно не может быть включено для отдельных ячеек. Однако, некоторые ячейки (или строки или столбцы) могут определяться как только для чтения при включенном редактировании для всей сетки. Вы можете определить, доступна ли сетка для редактирования методом IsEditable().

Вы можете установить состояние редактирования для определенной ячейки методом SetReadOnly(row, col, isReadOnly=True). Если параметр isReadOnly – True, то ячейка только для чтения. Значение параметра False делает ячейку доступной для редактирования. Метод SetReadOnly() - аналог метода с тем же названием класса wx.grid.GridCellAttr. Другими словами, вы могли установить признак ячейки только для чтения вот так GetCellAttr(row, col).SetReadOnly(isReadOnly). Преимущество использования механизма атрибутов ячейки состоит в том, что вы можете использовать методы SetRowAttr() и SetColAttr(), чтобы установить целые строки или столбцы как доступные для редактирования или только для чтения.

Вы можете также управлять редактором ячейки программно, используя методы EnableCellEditControl(enable=True) и DisableCellEditControl(), последний метод эквивалентен EnableCellEditControl(False). Метод EnableCellEditControl создаст и покажет редактор в текущей ячейке. Метод DisableCellEditControl скроет редактор в текущей ячейке, сохраняя отредактированные данные. Метод CanEnableCellControl() возвращает True, если сетка является доступной для редактирования и ячейка, не имеет признак только для чтения. Метод IsCellEditControlEnabled() возвращает True, если редактор ячейки активен.

Есть также некоторые методы для внутреннего использования, которые позволяют тонко управлять процессом редактирования. Вы можете вызвать редактирование текущей ячейки, используя метод ShowCellEditControl(), и закончить редактирование методом HideCellEditControl(). Вы можете определить, доступна ли текущая ячейка для редактирования, используя метод IsCurrentCellReadOnly(). Вы можете гарантировать, что новое значение будет сохранено, используя метод SaveEditControlValue(). Сетка неявно вызывает этот метод, когда текущая ячейка теряет фокус ввода, но это хороший метод для обработки ситуаций, которые могут вызвать потерю отредактированных данных (например, закрытие окна, которое содержит сетку).

Каждая отдельная ячейка имеет свой собственный объект редактора. Вы можете получить ссылку на этот объект редактора методом GetCellEditor(col, row), который возвращает объект класса wx.grid.GridCellEditor. Вы можете установить редактор методом

SetCellEditor(row, col, editor), где параметр editor – объект класса wx.grid.GridCellEditor. Вы можете управлять редактором по умолчанию для всей сетки методами GetDefaultEditor() и SetDefaultEditor(editor). Точно так же как renderers, объект редактора хранится в объекте wx.grid.GridCellAttr, связанном с ячейкой, строкой, или столбцом.

14.3.3 Как использовать редактор ячейки?

Как и в случае renderers, wxPython предлагает нескольким стандартных редакторов для различных типов данных, и дает вам возможность определить собственный редактор.

Предопределенные редакторы

Все wxPython редакторы – это подклассы класса wx.grid.GridCellEditor. В таблице 14.5 дано описание стандартных редакторов.

Таблица 14.5 Редакторы ячейки в wxPython

Редактор	**Описание**
wx.grid.GridCellAutoWrapStringEditor	Использует многострочный редактор для редактирования текстовых данных
wx.grid.GridCellBooleanEditor	Редактор для Булевских данных ячейки, представляет собой переключатель, который пользователь может включить или выключить. Визуально он немного отличается от переключателя, используемого Булевским renderer. Вы не обязаны использовать Булевский renderer, чтобы использовать Булевский редактор. Вы можете отображать данные как 1 или 0 или вкл\выкл.
wx.grid.GridCellChoiceEditor	Редактор для определенного списка опций. При вызове, пользователь видит выпадающий список для выбора. Конструктор принимает параметры (choices, allowOthers=False). Параметр choices - список строк. Если allowOthers - True, тогда пользователь может также напечатать произвольную строку в дополнение к списку.
wx.grid.GridCellEnumEditor	Является производным от wx.grid.GridCellChoiceEditor и управляет соответствием числового значения данных и строки, представленной пользователю.
wx.grid.GridCellFloatEditor	Редактор для ввода чисел с плавающей запятой и определенной точностью. Конструктор принимает параметры (width=-1, precision=-1), где width - минимальное число цифр, чтобы показать все число, и precision - максимальное число цифр после десятичной точки. Числа, введенные этим редактором, преобразуются к соответствующей ширине и точности.
wx.grid.GridCellNumberEditor	Редактор для ввода целых чисел. Конструктор принимает параметры (min=-1, max=-1). Если минимум и максимум установлены, редактор делает проверку диапазона и запрещает вводить числа вне этого диапазона. Если редактор с проверкой диапазона, он использует spinner control, чтобы пользователь мог изменять значения мышью.
wx.grid.GridCellTextEditor	Редактор по умолчанию для ввода строк.

В следующем разделе, мы покажем, как создать собственный редактор ячейки.

Создание собственного редактора ячейки

Вы может понадобиться собственный редактор, чтобы сделать свою обработку введенного значения. Для создания собственного редактора, создайте подкласс класса wx.grid.PyGridCellEditor. Иерархия классов редактора более сложна, чем иерархия renderer. В таблице 14.6 показано несколько методов, которые вы должны переопределить.

Таблица 14.6 Методы PyGridCellEditor, которые вы должны переопределить

Метод	**Описание**
BeginEdit(row, col, grid)	Параметры row и col - координаты ячейки, и grid – сетка, содержащая ячейку. Этот метод вызывается в начале редактирования. В этом методе осуществляется подготовка к редактированию.
Clone()	Возвращает копию редактора.
Create(parent, id, evtHandler)	Этот метод создает элемент управления, используемый редактором. Параметр parent - содержащий виджет, id – идентификатор, и evtHandler - обработчик событий, связанный с новым элементом управления.
EndEdit(row, col, grid)	Возвращает True, если редактирование изменило значение ячейки. Также здесь производится любая другая необходимая очистка.
Reset()	Вызывается при отмене редактирования. Должен восстановить первоначальное значение.

В таблице 14.7 показаны дополнительные методы родительского класса, который вы можете переопределить, чтобы улучшить поведение или вид вашего редактора.

Таблица 14.7 Методы PyGridCellEditor, которые вы можете переопределить

Метод	**Описание**
Destroy()	Выполняет заключительную очистку, при удалении объекта редактора.
IsAcceptedKey(evt)	Возвращает True, если клавиша, нажатая в evt используется для запуска редактора. Клавиша F2 всегда запускает редактор. Версия базового класса предполагает, что нажатие любой алфавитно-цифровой клавиши запустит редактор.
PaintBackground(rect, attr)	Принимает два параметра. Параметр rect типа wx.Rect с экранными координатами элемента управления, и параметр attr типа wc.grid.GridCellAttr – атрибуты, связанные с ячейкой. Цель этого метода состоит в том, чтобы рисовать любую часть ячейки, не охваченной редактированием непосредственно. Версия базового класса берет цвет фона из атрибутов и заполняет прямоугольник этим цветом.
SetSize(rect)	Параметр rect – объект wx.Rect с экранными координатами. Используйте этот метод если нужно, чтобы установить редактор в пределах прямоугольника.
Show(show, attr)	Параметр show - Булевское признак показывающий, должен ли редактор быть отображен, параметр attr – объект атрибутов ячейки. Метод вызывается, чтобы показать или скрыть элемент управления.
StartingClick()	Когда редактор вызывается щелчком мыши, этот метод вызывается, чтобы позволить редактору использовать этот щелчок в собственных целях.
StartingKey(evt)	Если редактор запущен нажатием клавиши, этот метод вызывается, чтобы позволить редактору использовать эту клавишу, если нужно.

Когда вы создадите свой редактор, вы можете установить его как редактор для любой ячейки, используя метод SetCellEditor. В листинге 14.8 показан простой редактор, который автоматически преобразует текст, который вы вводите в верхний регистр.

Листинг 14.8 Создание редактора для ввода символов в верхнем регистре

```
import wx
import wx.grid
import string

class UpCaseCellEditor(wx.grid.PyGridCellEditor):
    def __init__(self):
        wx.grid.PyGridCellEditor.__init__(self)

    def Create(self, parent, id, evtHandler):
        """
        Called to create the control, which must derive from wx.Control.
        *Must Override*
        """
        self._tc = wx.TextCtrl(parent, id, "")
        self._tc.SetInsertionPoint(0)
        self.SetControl(self._tc)

        if evtHandler:
            self._tc.PushEventHandler(evtHandler)

        self._tc.Bind(wx.EVT_CHAR, self.OnChar)

    def SetSize(self, rect):
        """
        Called to position/size the edit control within the cell rectangle.
        If you don't fill the cell (the rect) then be sure to override
        PaintBackground and do something meaningful there.
        """
        self._tc.SetDimensions(rect.x, rect.y, rect.width+2, rect.height+2,
                               wx.SIZE_ALLOW_MINUS_ONE)

    def BeginEdit(self, row, col, grid):
        """
        Fetch the value from the table and prepare the edit control
        to begin editing.  Set the focus to the edit control.
        *Must Override*
        """
        self.startValue = grid.GetTable().GetValue(row, col)
        self._tc.SetValue(self.startValue)
        self._tc.SetInsertionPointEnd()
        self._tc.SetFocus()
        self._tc.SetSelection(0, self._tc.GetLastPosition())

    def EndEdit(self, row, col, grid):
        """
        Complete the editing of the current cell. Returns True if the value
        has changed.  If necessary, the control may be destroyed.
        *Must Override*
        """
        changed = False

        val = self._tc.GetValue()

        if val != self.startValue:
            changed = True
            grid.GetTable().SetValue(row, col, val) # update the table

        self.startValue = ''
        self._tc.SetValue('')
        return changed
```

```
    def Reset(self):
        """
        Reset the value in the control back to its starting value.
        *Must Override*
        """
        self._tc.SetValue(self.startValue)
        self._tc.SetInsertionPointEnd()

    def Clone(self):
        """
        Create a new object which is the copy of this one
        *Must Override*
        """
        return UpCaseCellEditor()

    def StartingKey(self, evt):
        """
        If the editor is enabled by pressing keys on the grid, this will be
        called to let the editor do something about that first key if
        desired.
        """
        self.OnChar(evt)
        if evt.GetSkipped():
            self._tc.EmulateKeyPress(evt)

    def OnChar(self, evt):
        key = evt.GetKeyCode()
        if key > 255:
            evt.Skip()
            return
        char = chr(key)
        if char in string.letters:
            char = char.upper()
            self._tc.WriteText(char)
        else:
            evt.Skip()

class TestFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, title="Grid Editor",
                          size=(640,480))

        grid = wx.grid.Grid(self)
        grid.CreateGrid(50,50)
        grid.SetDefaultEditor(UpCaseCellEditor())


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

Для справки по методам, используемым в классе редактора, обращайтесь к таблицам 14.6 и 14.7.

14.4 Обработка событий

Сетка создает множество событий, которые вы можете обработать. Мы отделим события клавиатуры и события мыши. Это позволит более точно настроить реакцию вашей сетки на события.

14.4.1 Обработка событий мыши

Для сетки, есть несколько различных типов событий мыши, и есть также несколько различных классов событий для этих типов. Обычно используемый класс события - wx.grid.GridEvent. Класс события сетки – это подкласс wx.CommandEvent. Он обеспечивает несколько методов, чтобы получить детальную информацию о событии, как отображено в таблице 14.8.

Table 14.8 Методы wx.grid.GridEvent

Метод	**Описание**
AltDown()	Возвращает True, если при вызове события клавиша alt была нажата.
ControlDown()	Возвращает True, если при вызове события клавиша control была нажата.
GetCol()	Возвращает индекс столбца ячейки, где произошло событие.
GetPosition()	Возвращает объект wx.Point, представляющий координаты в пикселях точки, где произошло событие.
GetRow()	Возвращает индекс строки ячейки, где произошло событие.
MetaDown()	Возвращает True, если при вызове события клавиша meta была нажата.
Selecting()	Возвращает True, если событие – выбор ячейки, и False, если событие - отмена выбора ячейки.
ShiftDown()	Возвращает True, если при вызове события клавиша shift была нажата.

Есть несколько других типов событий, связанных с wx.grid.GridEvent. Они перечислены в таблице 14.9.

Таблица 14.9 Типы событий мыши для ячейки сетки

Тип события	**Описание**
wx.grid.EVT_GRID_CELL_CHANGE	Возникает, когда пользователь изменяет данные в ячейке через редактор.
wx.grid.EVT_GRID_CELL_LEFT_CLICK	Возникает, когда пользователь выполняет щелчок левой кнопкой мыши в ячейке.
wx.grid.EVT_GRID_CELL_LEFT_DCLICK	Возникает, когда пользователь выполняет двойной щелчок левой кнопкой мыши в ячейке.
wx.grid.EVT_GRID_CELL_RIGHT_CLICK	Возникает, когда пользователь выполняет щелчок правой кнопкой мыши в ячейке.
wx.grid.EVT_GRID_CELL_RIGHT_DCLICK	Возникает, когда пользователь выполняет двойной щелчок правой кнопкой мыши в ячейке.
wx.grid.EVT_GRID_EDITOR_HIDDEN	Возникает, когда редактор ячейки скрыт в конце сеанса редактирования.
wx.grid.EVT_GRID_EDITOR_SHOWN	Возникает, когда редактор ячейки показывается в начале сеанса редактирования.
wx.grid.EVT_GRID_LABEL_LEFT_CLICK	Возникает, когда пользователь выполняет щелчок левой кнопкой мыши в области заголовков столбца или строки.
wx.grid.EVT_GRID_LABEL_LEFT_DCLICK	Возникает, когда пользователь выполняет двойной щелчок левой кнопкой мыши в области заголовков столбца или строки.
wx.grid.EVT_GRID_LABEL_RIGHT_CLICK	Возникает, когда пользователь выполняет щелчок правой кнопкой мыши в области заголовков столбца или строки.
wx.grid.EVT_GRID_LABEL_RIGHT_DCLICK	Возникает, когда пользователь выполняет двойной щелчок правой кнопкой мыши в области заголовков столбца или строки.
wx.grid.EVT_GRID_SELECT_CELL	Возникает, когда пользователь

	перемещает курсор в новую ячейку, выбирая ее.

Есть еще два типа события wx.grid.GridSizeEvent. Тип события - wx.grid.EVT_GRID_COL_SIZE, возникает, когда изменяется размер столбца, и wx.grid.EVT_GRID_ROW_SIZE, возникает, когда изменяется размер строки. Событие размера сетки имеет пять тех же самых методов как и wx.GridEvent - AltDown(), ControlDown(), GetPosition(), MetaDown() и ShiftDown(). Есть еще метод GetRowOrCol(), который возвращает индекс строки или столбца, в зависимости от типа события.

Есть еще событие wx.grid.GridRangeSelectEvent. Тип события - wx.grid.EVT_GRID_RANGE_SELECT. Оно возникает, когда пользователь выбирает непрерывный блок ячеек. Объект события имеет методы GetBottomRightCoords(), GetBottomRow(), GetLeftCol(), GetRightCol(), GetTopRightCoords() и GetTopRow(). Они возвращают соответствующие индексы строки и столбца выбранного блока. Возвращаемым значением для координатных методов является кортеж (row, col).

И наконец, событие wx.grid.GridEditorCreatedEvent с типом события EVT_GRID_EDITOR_CREATED. Название подразумевает, что событие возникает, когда редактор создан сеансом редактирования. Объект события имеет методы GetCol(), GetRow() и GetControl(), которые возвращают индекс столбца, индекс строки и используемый элемент управления редактирования, соответственно.

14.4.2 Обработка событий клавиатуры

В дополнение к мыши, для навигации по сетке пользователь может использовать клавиатуру. Вы также можете программно изменить положение курсора методами перемещения, перечисленными в таблице 14.10. Многие из методов принимают параметр expandSelection. Этот параметр работает одинаково в каждом методе. Если параметр - True, текущий выбор будет изменен так, чтобы включить новую позицию курсора. Если параметр - False, текущий выбор будет заменен новым курсором.

Таблица 14.10 Методы перемещения курсора сетки

Метод	**Описание**
MoveCursorDown(expandSelection)	Перемещает курсор вниз. Метод эквивалентен нажатию клавиши «стрелка вниз» (если expandSelection=False) или shift-стрелка вниз (если expandSelection=True).
MoveCursorDownBlock(expandSelection)	Перемещает курсор на одну ячейку ниже выбранного блока. Метод эквивалентен нажатию ctrl-down (если expandSelection=False) или shift-ctrl-down (если expandSelection=True).
MoveCursorLeft(expandSelection)	Перемещает курсор влево. Метод эквивалентен нажатию клавиши «стрелка влево» (если expandSelection=False) или shift-стрелка влево (если expandSelection=True).
MoveCursorLeftBlock(expandSelection)	Перемещает курсор на одну ячейку левее выбранного блока. Метод эквивалентен нажатию ctrl-left (если expandSelection=False) или shift-ctrl-left (если expandSelection=True).
MoveCursorRight(expandSelection)	Перемещает курсор вправо. Метод эквивалентен нажатию клавиши «стрелка вправо» (если expandSelection=False) или shift-стрелка вправо (если expandSelection=True).
MoveCursorRightBlock(expandSelection)	Перемещает курсор на одну ячейку правее выбранного блока. Метод эквивалентен

	нажатию ctrl-right (если expandSelection=False) или shift-ctrl-right (если expandSelection=True).
MoveCursorUp(expandSelection)	Перемещает курсор вверх. Метод эквивалентен нажатию клавиши «стрелка вверх» (если expandSelection=False) или shift-стрелка вверх (если expandSelection=True).
MoveCursorUpBlock(expandSelection)	Перемещает курсор на одну ячейку выше выбранного блока. Метод эквивалентен нажатию ctrl-up (если expandSelection=False) или shift-ctrl-up (если expandSelection=True).
MovePageDown()	Перемещает курсор вниз так, что ячейки нижней строки перемещаются в верхнюю строку.
MovePageUp()	Перемещает курсор вверх так, что ячейки верхней строки перемещаются в нижнюю строку.

Мы рассказали почти все из того, что вы должны знать о сетках. В следующей главе, мы займемся следующим виджетом – деревом (tree control).

14.5 Резюме

- Сетка позволяет создать элемент управления подобный электронной таблице. Сетка - объект класса wx.grid.Grid. Обычно сетки достаточно сложны, поэтому лучше создать ваш собственный подкласс сетки с его собственным методом __init__ вместо того, чтобы создавать объект базового класса, и вызывать его методы в другом месте вашего приложения.

- Есть два способа заполнить сетку данными. Данные можно добавить явно, инициализировав сетку методом CreateGrid(numRows, numCols), после чего установить значения отдельных ячеек методом SetCellValue(row, col, s). Другой способ: вы можете создать объект таблицы сетки, который позволит использовать данные из другого источника для отображения в сетке. Таблица сетки – это подкласс wx.grid.PyGridTableBase с методами, типа GetValue(row, col), которые нужно переопределить, чтобы управлять поведением сетки и отображением ячеек. Таблица связывается с сеткой методом SetTable(table). Когда сетка связана с таблицей, именно таблица управляет изменениями размера сетки, добавлением строк и столбцов, и методами удаления.

- Сетка имеет заголовки строк и столбцов, которые по умолчанию имеют значения, подобные тем, что вы ожидали бы в электронной таблице. Текст заголовков и другие атрибуты могут быть изменены свойствами сетки. Размер строки или столбца может быть установлен явно для каждого элемента, или сетка может установить размер элементов автоматически, основываясь на отображаемых данных. Пользователь может также изменять размер элементов сетки, перетаскивая линии сетки. Вы можете установить минимальный размер для каждой строки или столбца, если хотите запретить ячейкам становиться слишком маленькими. Кроме того, определенные ячейки могут заполнять другие строки или столбцы, используя метод SetCellSize(row, col, numrows, numcols).

- Пользователь может выбрать один или более блоков ячеек в сетке. Эти же действия можно выполнить программно множеством различных методов выбора. Ячейка сетки, которая скрыта прокруткой, может быть отображена методом MakeCellVisible(row, col).

- Большая часть мощи и гибкости сетки исходит из способности создавать собственные renderers (способы отображения) и редакторы для каждой ячейки.

Renderer управляет отображением информации в ячейке. Renderer по умолчанию - простая строка, но есть предопределенные renderers для Булевских, целочисленных, и данных с плавающей запятой. Вы можете создать ваш собственный renderer как подкласс класса wx.Grid.PyGridCellRenderer, переопределив соответствующие методы.

- По умолчанию, сетка позволяет редактирование данных непосредственно в ячейке. Вы можете изменить это поведение для отдельной ячейки, строки, столбца, или для всей сетки. Объект редактора управляет редактирование ячейки. Есть предопределенные редакторы для Булевских, целочисленных, данных с плавающей запятой и для списков. Вы можете создать ваш собственный редактор как подкласс класса wx.grid.GridCellEditor, переопределив несколько методов базового класса.

- Сетка имеет множество событий, которые вы можете перехватить и обработать, включая события для щелчков мыши в ячейках и в заголовках, и события, вызванные изменением размера ячейки. Кроме того, вы можете программно вызвать методы навигации курсора в сетке.