目录

1. WinForms 平台中 FlexGrid 的快速启动

在本节中，你将学到如下应用，即如何使用基本的 C1FlexGrid 功能来创建一个简单的表格。本节并不是一个关于 C1FlexGrid 所有功能的全面教程，而是提供一种快速启动并突出强调一些常规的方法来使用这个产品。至于更深入的教程，请参阅 FlexGrid for WinForms 教程。本例中使用 Visual Studio 2010，它的操作步骤可能与其他版本的 Visual Studio 中的略有不同。

1.1 第一步：为 WinForms 应用程序创建 FlexGrid

以下步骤将引导你创建一个简单的表格应用。

1. 创建一个新项目。
2. 在窗体上添加一个 C1FlexGrid 控件。
3. 打开 C1FlexGrid 任务菜单。有关如何访问 C1FlexGrid 任务菜单的详细信息请参阅 C1FlexGrid 任务菜单。
4. 在 C1FlexGrid 任务菜单中，单击“停靠在父容器”。这个操作将表格中 “停靠”的属性设置为“填充”，设置后这个表格就会充满整个窗体。

运行该程序。一个简单的表格应用将会出现。

恭喜你！你已经成功地创建了一个简单的表格应用。在下一个主题中，你将会学到如何将 C1FlexGrid 控件绑定到一个数据源上。

1.2 第二步：将 C1FlexGrid 控件绑定到一个数据源

上文中，即“第一步：为 WinForms 应用程序创建 FlexGrid”你已经创建了一个表格应用。以下步骤将引导你实现一个数据源与这个表格应用的绑定。

1. 打开 **C1FlexGrid 任务菜单**。有关如何访问 C1FlexGrid 任务菜单的详细信息请参阅 **C1FlexGrid 任务菜单**。
2. 在 C1FlexGrid 任务菜单中，单击“**选择数据源**”下拉箭头并从下拉框中选择“**添加项目数据源**”链接。
3. “**数据源配置向导**”出现。保留默认设置，在“**选择数据源类型**”页面上选择“**数据库”**，然后单击“**下一步**”。
4. 在“**选择一个数据库模型**”页面上，仍然保持选择“**数据库**”状态，然后单击“**下一步”**。
5. 单击“**新的连接**”按钮来创建一个新的连接，或从下拉列表中选择一种。当你单击“**新的连接**”时，“**添加连接**”对话框出现。
6. 保留“Microsoft Access 数据库文件”作为“数据源”。
7. 单击在**“数据库文件名**”下面的“**浏览**”按钮。在“**选择 Microsoft Access 数据库文件**”对话框中，浏览如下地址 C:\Documents and Settings\<用户名>\My Documents\ComponentOne Samples\Common (Windows XP) or C:\Users\<用 户 名>\Documents\ComponentOne

Samples\Common (Vista)目录中的 NWind.mdb 数据库。选择 NWind.mdb 文件，并单击“**打开**”。

8. 在“**添加连接**”的对话框中，单击“**测试连接**”按钮以便确认你已经成功地连接上了数据库或服务器，然后单击“**确定**”。
9. 再一次单击“**确定**”来关闭“**添加连接**”的对话框。
10. 单击“**下一步**”按钮来继续操作。会出现一个对话框来询问你是否想添加这个数据库文件到你的项目并修改连接字符串。如果不需要复制这个数据库到你的项目，请单击“**否**”。
11. 在此应用程序的配置文件中通过选择“**是的，将此连接另存为**.....”来保存连接字符串，并输入一个名称。单击“**下一步**”按钮来继续操作。
12. 在“**选择数据库对象**”页面上，展开“**表格**”节点，并选择“**产品**”表。在“**数据集名称**”栏中输入“**产品数据集**”，之后单击“**完成**”来退出向导。
13. 这样，一个数据集和连接字符串就被添加到你的项目里了。除此之外，Visual Studio 会自动创建以下代码来填充这个数据集：

- Visual Basic

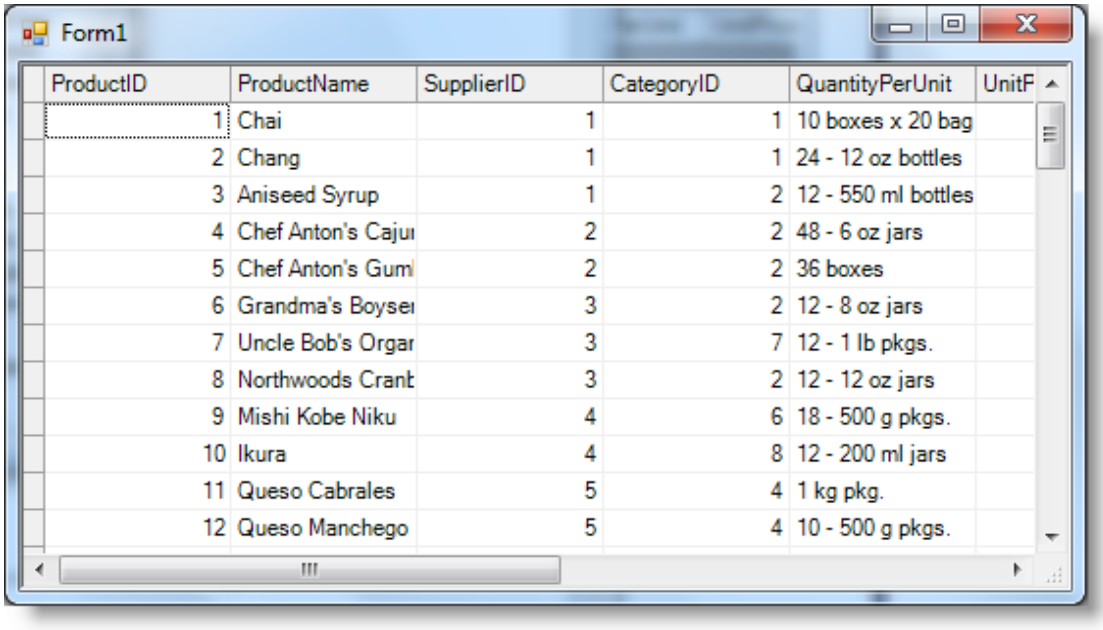
```
Me.ProductsTableAdapter.Fill(Me.ProductsDS.Products)
```

- C#

```
this.productsTableAdapter.Fill(this.productsDS.Products);
```

运行该程序并遵守以下规定：

请注意，“产品”表中的数据会反映在表格

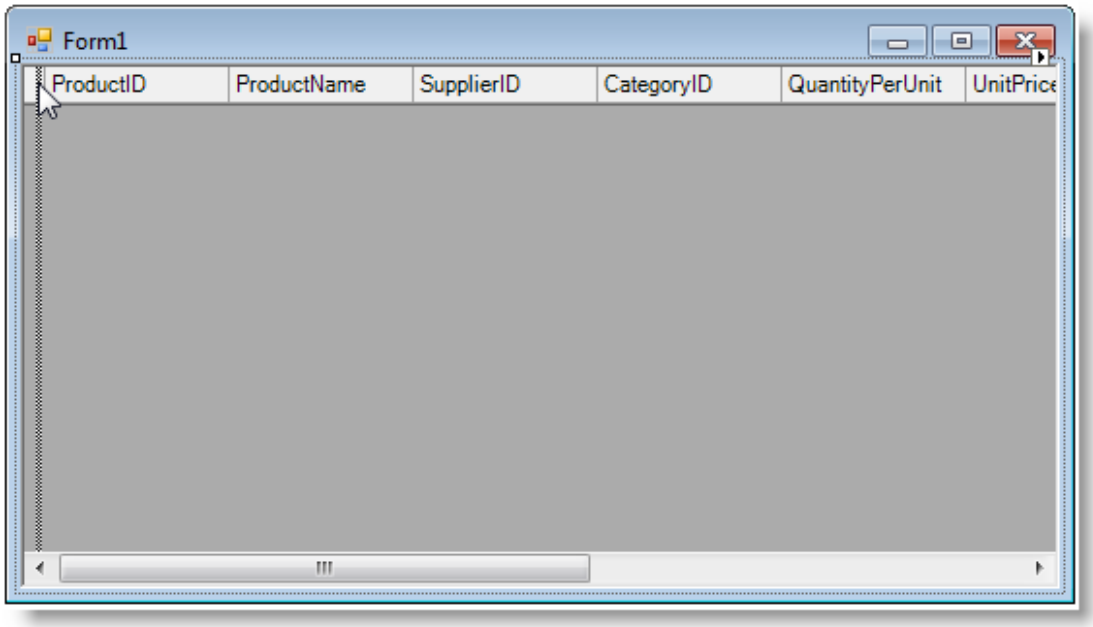


恭喜你！你已经将一个表格应用成功地绑定在了一个数据源上。在下一个标题中，你将会学到如何自定义格式字符串，视觉样式和内置样式。

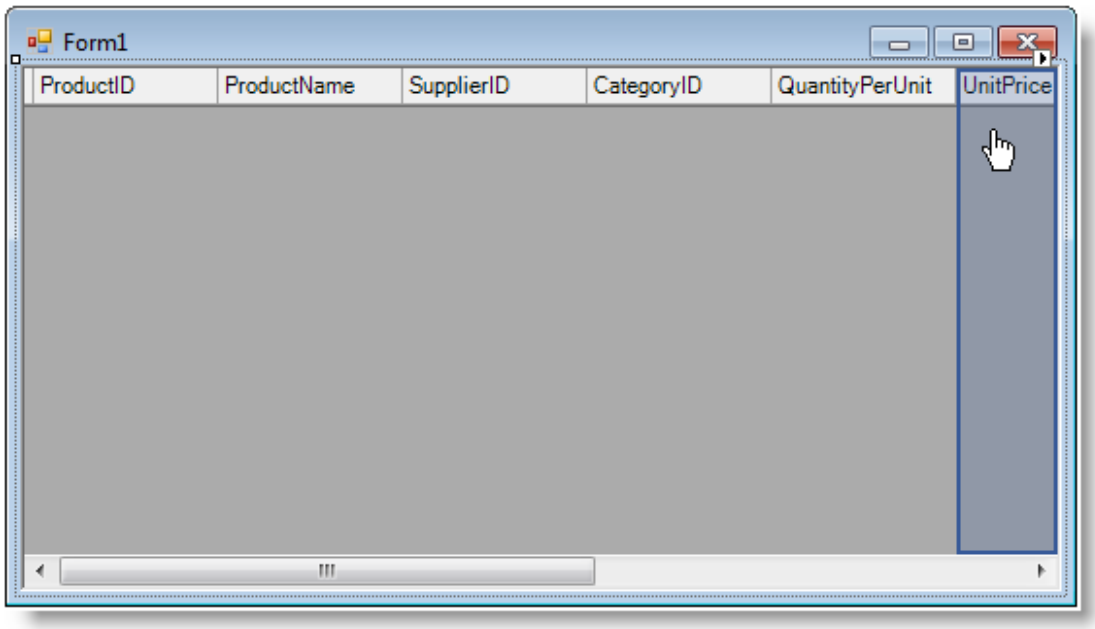
1.3 第三步：自定义 C1FlexGrid 设置

以下步骤将引导你实现表格中格式字符串，视觉样式和内置样式的设置。

1. 向左拖动出现在第一列标题右边缘的水平双箭头，直到可以看见“单价”这一列，以此来调整表格中的第一列。



2. 单击“单价”这一列来打开“列任务”菜单。

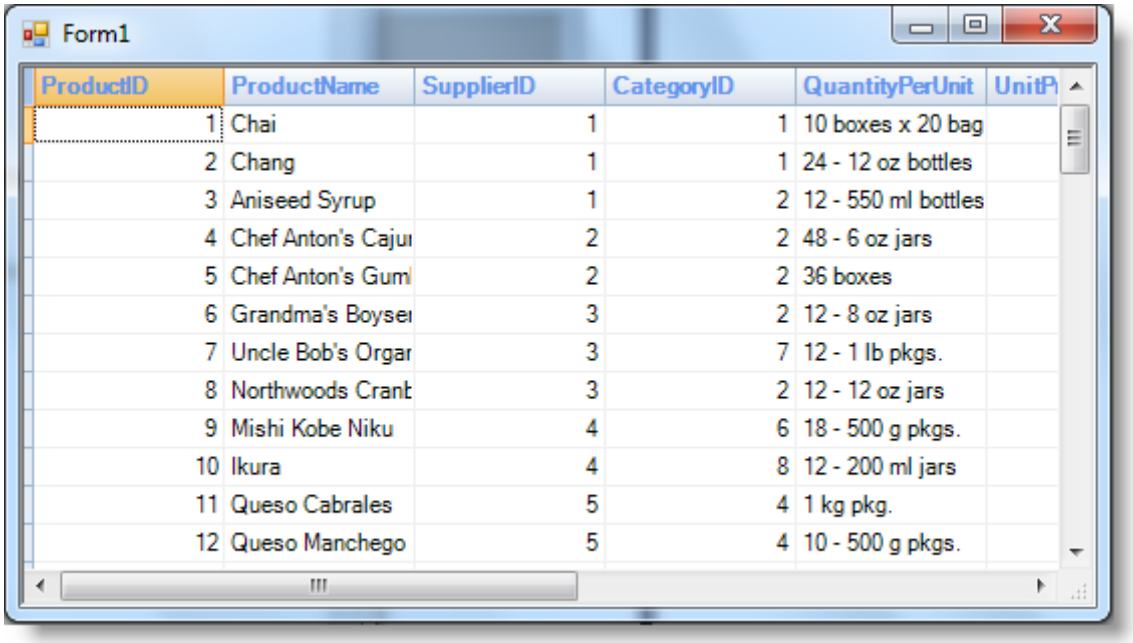


3. 单击 “格式字符串”文本框旁边的“省略号”按钮来打开“格式字符串”对话框。
4. 在“格式字符串”对话框中，选择“格式类型”下面的“货币”。然后，单击“确定”。

5. 在属性窗口中，找到“视觉样式”属性，并将其设置为“Office2007Blue”。
6. 打开 C1FlexGrid 任务菜单。由于我们上次曾经使用“任务”菜单来编辑了一列，“列任务”菜单就会出现。选择 C1FlexGrid 任务菜单来返回到 C1FlexGrid 任务菜单。
7. 在 C1FlexGrid 任务菜单上，选择“样式”来打开 C1FlexGrid 样式编辑器（第 27 页）。
8. 在“内置样式”下面的“C1FlexGrid 样式编辑器”中，选择“已修订”。
9. 在右窗格中展开“字体”节点，并且把“粗体”的属性设置为“真”。
10. 在“网站”标签上将“前景色”的属性设置为“矢车菊蓝”，然后单击“确定”来关闭对话框。

运行该程序并遵守以下规定：

这个表格的应用程序就会用格式字符串，视觉样式和内置样式来显示一个产品表。



恭喜你！你已经成功地设置了一个表格的格式字符串，视觉样式，以及内置样式。关于快速启动的介绍到此结束。

2. 设计时支持

在设计时使用 Visual Studio 中的属性表格，菜单和设计器，你可以轻松地实现 **ComponentOne FlexGrid for WinForms** 配置。以下各节描述了如何使用 C1FlexGrid 的设计时环境来配置 C1FlexGrid 控件。

2.1 C1FlexGrid 编辑器

C1FlexGrid 列编辑器和 **C1FlexGrid 样式编辑器**，利用这两个设计时编辑器，你可以控制 C1FlexGrid 的布局和外观。除此之外，利用标题样式和列样式这两个设计时编辑器你可以改变一个特定的标题或列的外观。

2.1.1 C1FlexGrid 列编辑器

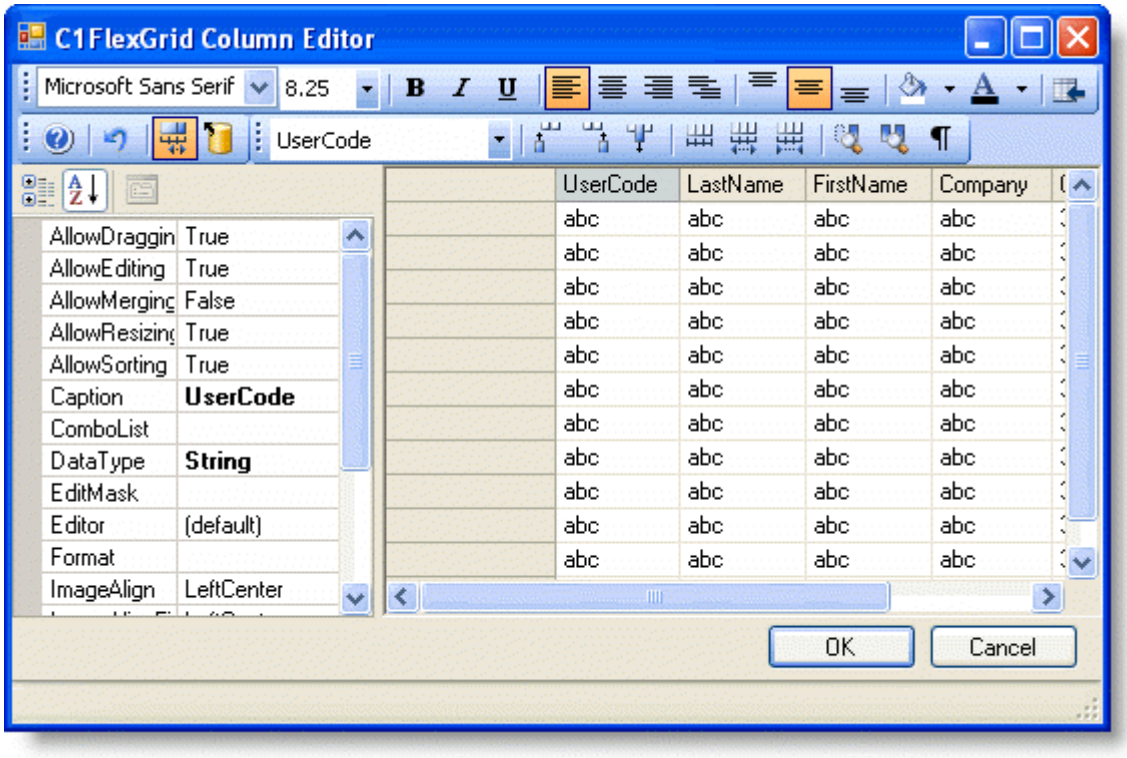
只要你愿意，你就可以在设计时就设置好表格列，而不是靠编写代码来执行这一步操作。要访问设计器你可以通过以下三种途经中的任意一种：

在“设计”视图中选择表格，进入“**属性**”窗口，然后单击一下 COLS 属性旁边的省略号按钮（...）。

右键单击此控件，并在上下文菜单中选择“**设计器**”。

单击表格右上角的智能标签（ ），并从 C1FlexGrid 任务菜单中选择“**设计器**”。

这将使列编辑器显示如下图所示：

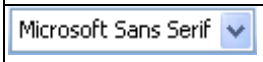
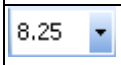


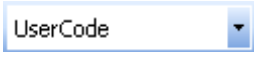
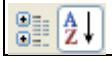
在绑定模式下，编辑器可用于选择应显示在数据源中的如下区域，它们的

顺序，列标题，宽度和对齐。在未绑定模式下，编辑器也可以用来选择列数据类型。

编辑器允许您执行以下操作：

- **将列重新排序**：你可以通过用鼠标拖动标题单元格来将所选择的列移动到新的位置。
- **调整列宽**：你可以通过用鼠标拖动标题单元格的右边缘来调整列宽。你也可以通过按住 SHIFT 键并单击标题单元格来选择多列，然后，一次性地使用属性窗口来设置所有列的宽度。比如将设置为-1 的列宽度恢复到默认宽度。
- **设置列属性**：每当选择了一个或多个的列，你都可以在编辑器左侧的属性表格中看到并编辑其属性。
- **插入或删除列**：使用工具栏在所选位置之前或之后插入列（大多在绑定模式下比较实用），或删除列。
- **使用工具栏来执行常规任务**：以下列表分别描述了工具栏中各个按钮的功能：

按钮	描述
	**字体**：在下拉列表中设置所选定的字体。
	**字号**：在下拉列表中设置所选定的字体大小。
	**字体格式**：可将粗体，斜体或下划线这几种字体应用于所选文字。
	**字体水平对齐**：可将字体对齐设置为居左，居中，居右或常规。
	**字体垂直对齐**：将列的内容对齐到顶端，居中或底部。这些按钮仅仅会影响到表格的滚动区域。要想设置标题列的对齐的话，请选择列，并设置 TextAlignFixed 的属性。
	**背景色**：可以用于所选列的背景色的设置。
	**前景色**：可以用于所选列的前景色的设置。
	**适用于固定行**：可将此设置应

	用于固定行。
	**帮助**：用于显示或隐藏对所选定的属性的描述。
	**撤消**：取消所有更改，将表格列恢复到它们原本的状态。
	**自动调整**：当表格被绑定到一个数据源时，用它来判断表格是否应自动调整所有列来适应其内容。
	**从数据源上重新加载**：用从当前数据源上得来的信息来对所有列进行重置。当表格被绑定到一个数据源上并且你想要从头开始编辑时，这个按钮非常有用。当表格未被绑定到一个数据源上时，此按钮被禁用。
	**选定的列**：可用于从下拉列表中选择当前列。
	**插入列**：在所选定的位置左边或者右边插入列。
	**删除列**：删除所选列。
	**列的宽度**：将所选定的所有列的列宽设置为相同的宽度，较之以前更宽或更窄。
	**切换显示**：显示或隐藏列。
	**取消隐藏所有列**：使所有列可见。
	**显示隐藏列**：显示隐藏列。如果你将列的可见属性更改为“假”，它就会被隐藏，因此，你将无法用鼠标来选择它。这时候使用这个按钮就可以显示所有隐藏的列，以便你可以选择和编辑它们。
	用这个工具来判断所选定的列

	的属性应该按分类显示还是按字母顺序显示。

2.1.2 C1FlexGrid 样式编辑器

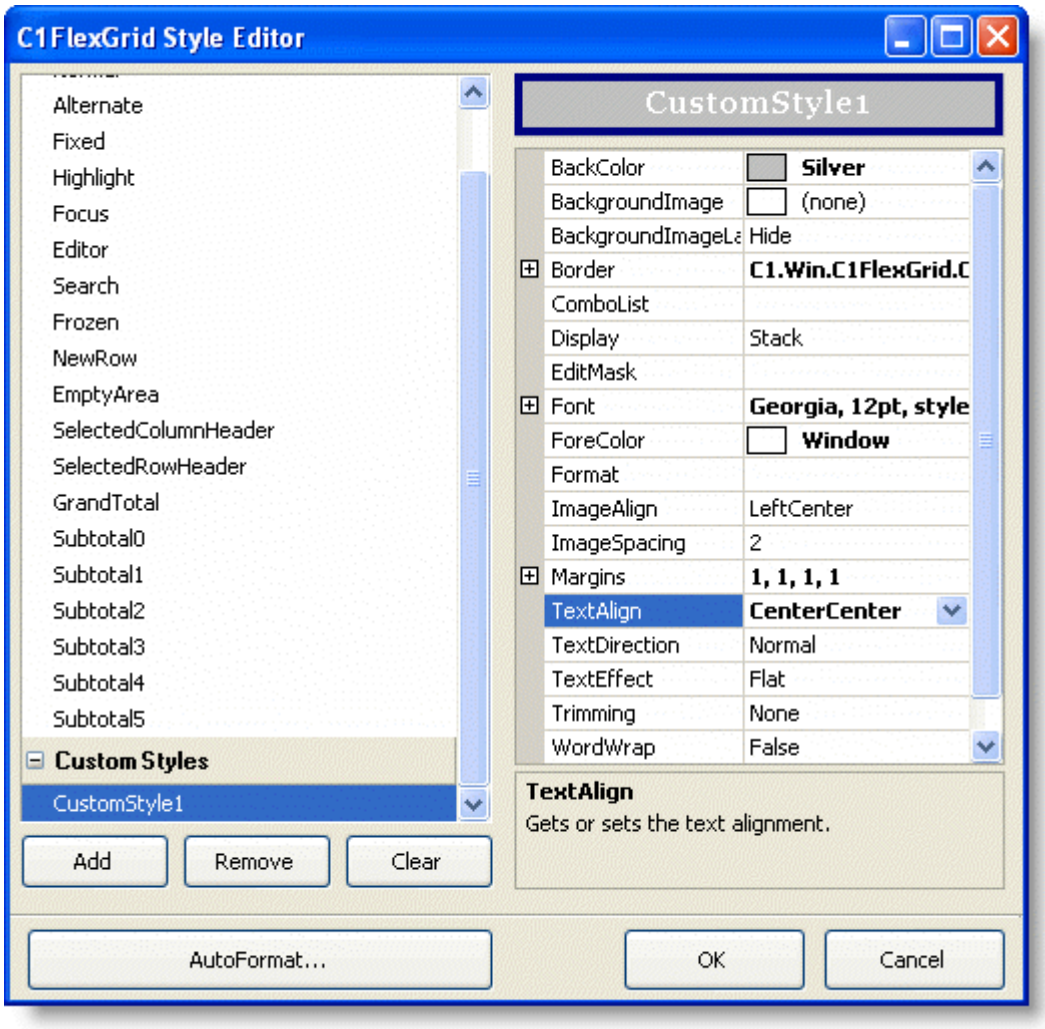
只要你愿意，你就可以在设计时就设置好样式，而不是靠编写代码来执行这一步操作。要访问 C1FlexGrid 样式编辑器你可以通过以下三种途经中的任意一种：

- 选择表格，进入“**属性**”窗口，然后单击一下样式属性旁边的省略号按钮。
- 右键单击表格，并从上下文菜单中选择样式。
- 单击表格右上角的智能标签（ ），并从 C1FlexGrid 任务菜单中选择样式。

这时，表格将会弹出 C1FlexGrid 样式编辑器的对话框。

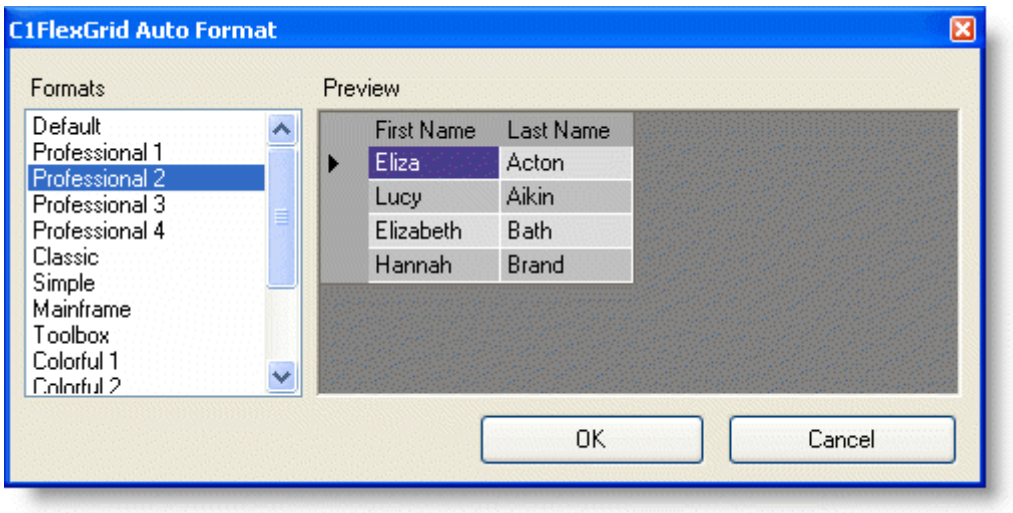
样式编辑器可以让你修改现有的样式，并添加一些新的、自定义的，这些以后也许会被分配到单元格、行和列。

使用“**添加**”按钮来添加一个自定义样式。你可以看到你新添加的样式是如何出现在样式属性上方的预览区域的。



“**移除**”按钮可以删除所选定的自定义样式。你可以通过在列表中选中它们并键入一个新的名称来重命名自定义样式。“**清除**”按钮可以删除所有的自定义样式，并且将内置样式恢复为其默认值。

使用“**自动套用格式**”按钮，会弹出一个二级对话框，你可以在其中选择一套完整的预定样式。以下就是“自动套用格式”对话框看起来的样子：



2.1.3 标题样式和列样式

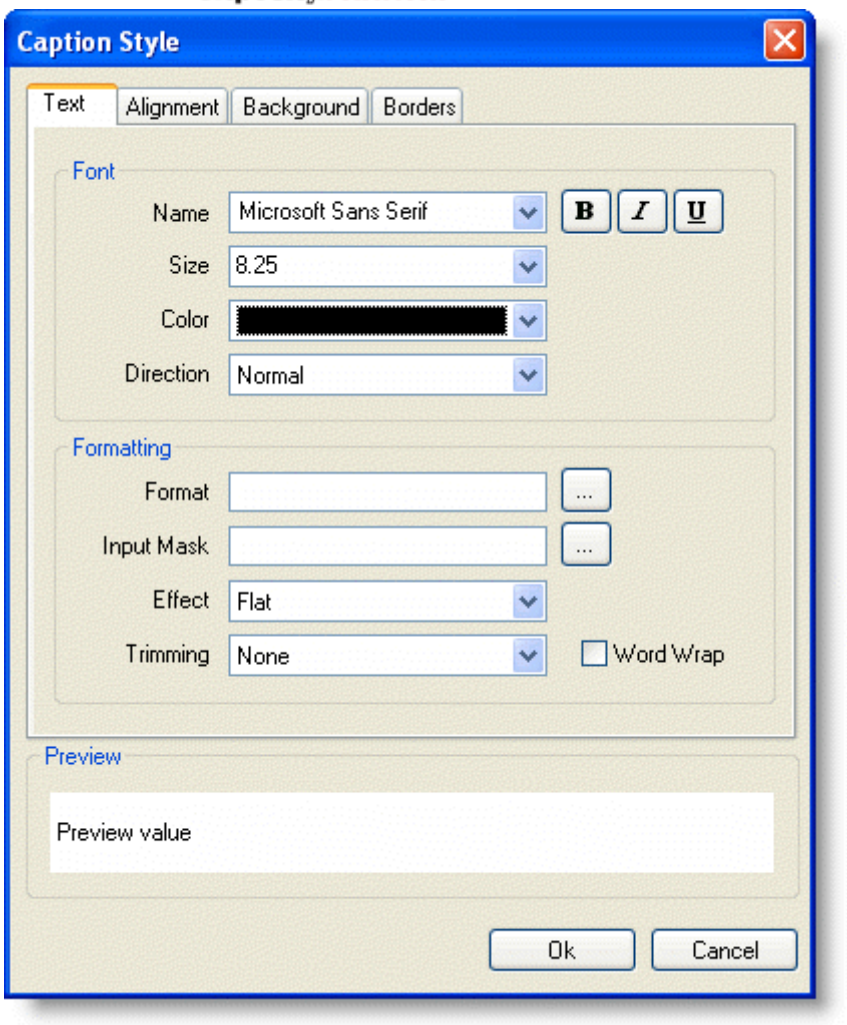
你可以使用**标题样式**编辑器和**列样式**编辑器，为一个选定的列来明确其标题文本和列中文本的属性，除此之外，还分别包括对齐，背景和边框。

标题样式和**列样式**编辑器只能通过“**列任务**”菜单来访问。如需“**列任务**”菜单上的更多信息，请参阅“列任务菜单”。

标题样式和**列样式**编辑器有四个标签：**文本**，**对齐**，**背景**和**边框**。在将所有设置应用于表格之前，你可以通过**预览**区域来查看你的设置。

文本

“文本”标签可以用来设置标题的字体和格式。



在**字体**方面可以用到下列选项：

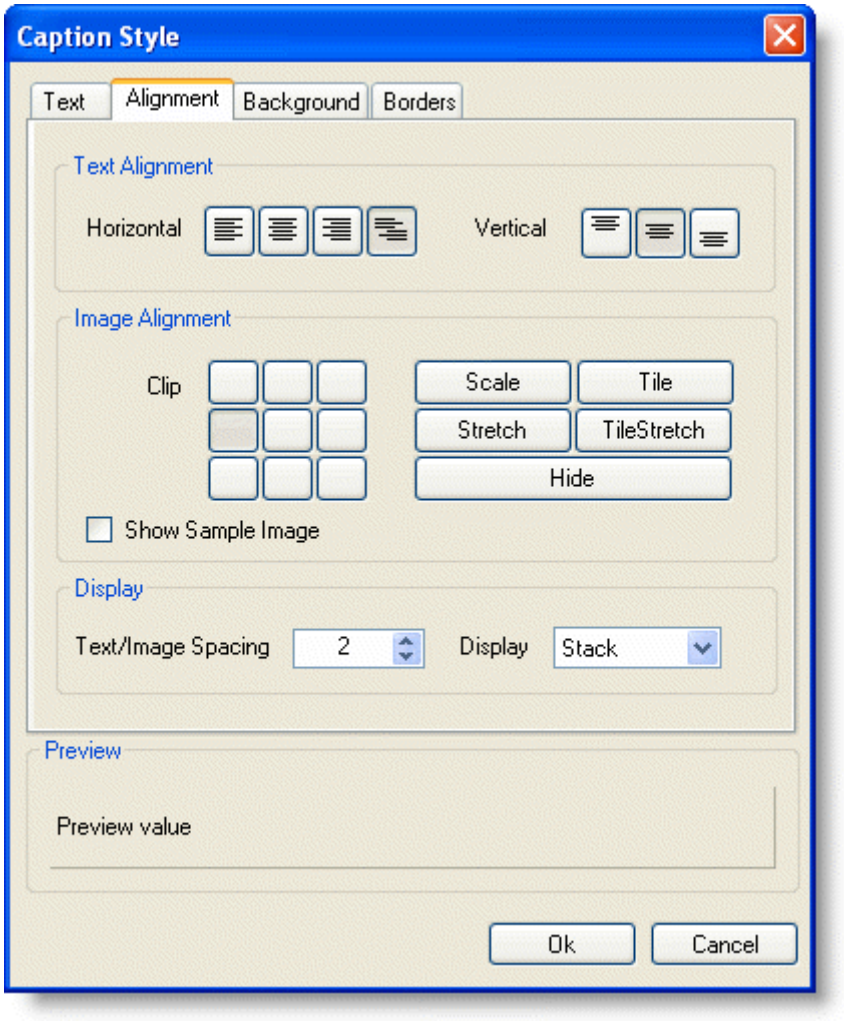
- **名称**：选择一个字体的名称。
- **尺寸**：选择一个字体大小。
- **颜色**：选择一个字体颜色。
- **方向**：从常规，向上或向下中选择。
- **字体效果**：使用按钮来进行粗体、斜体与下划线之间的切换，进行开启或关闭。

在格式方面可以用到下列选项：

- **格式**：单击省略号按钮来打开“格式字符串”对话框。有关“格式字符串”对话框的更多详细信息请参阅 “单元格的内容”（第 46 页）。
- **输入掩码**：单击省略号按钮打开“输入掩码”对话框。有关“输入掩码”对话框的更多详细信息请参阅 “掩码” （第 58 页）。
- **效果**：在“平面”、“凸起”或“嵌入”中选择。有关输入文本效果选项的更多详细信息请参阅“TextEffectEnum Enumeration”。
- **剪裁**：从“无”、“字符”、“文字”、“省略字符”、“省略文字”、“省略路径”中选择所需要的来设置长字符串应该如何调整以适应单元格。
- **自动换行**：勾选这一栏来允许标题中的文字自动换行。

对齐

对齐标签可以对标题中的文本和图像两方面进行对齐设置。



在**文本对齐**方面可以用到下列选项：

- **水平对齐**：单击以下按钮来进行不同对齐之间的切换，左对齐，居中对齐，右对齐和常规对齐。
- **垂直对齐**：单击以下按钮来进行不同对齐之间的切换，顶端对齐，中间对齐和底部对齐。
- 在图像对齐方面可以用到下列选项：
- **Clip**：点击那些按钮来进行单元格中图像对齐的切换，或者 Scale, Tile, Stretch, TileStretch, 或 Hide 该图像。有关图像对齐方面不同选项的更多详细信息请参阅“ImageAlignEnum Enumeration”。
- **显示样本图像**：勾选“显示样本图像”栏来在预览区域中显示样本图像。

在显示区域可以用到下列选项：

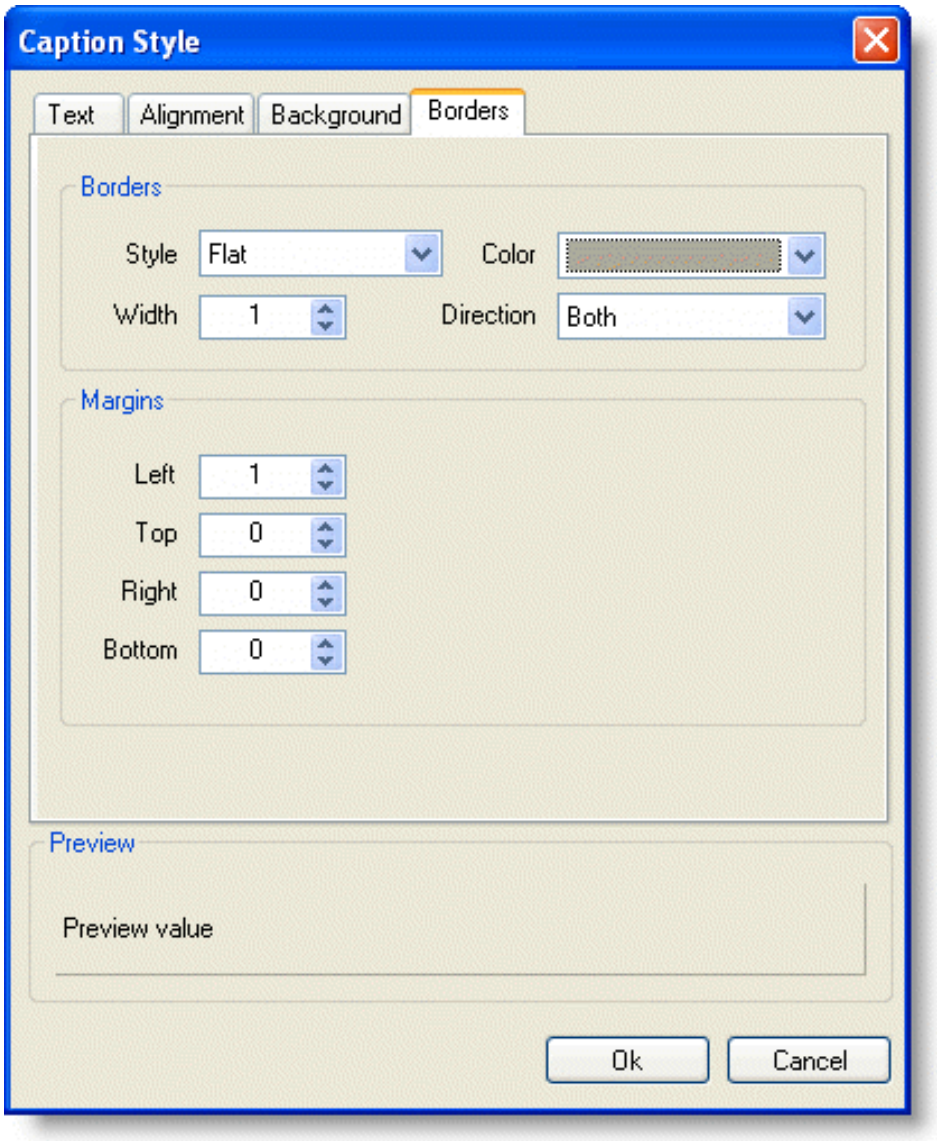
文本/图像间距：增加或减少该值来增加或减少文本和图像之间的间距量。

显示：在“纯文本”、“纯图像”、“覆盖”、“叠加”或“无”中选择

所需要的。有关显示的不同选项的更多详细信息请参阅“DisplayEnum Enumeration”。

背景

背景标签可以用来设置背景颜色和背景图片。



在**背景颜色**区域可以用到下列选项：

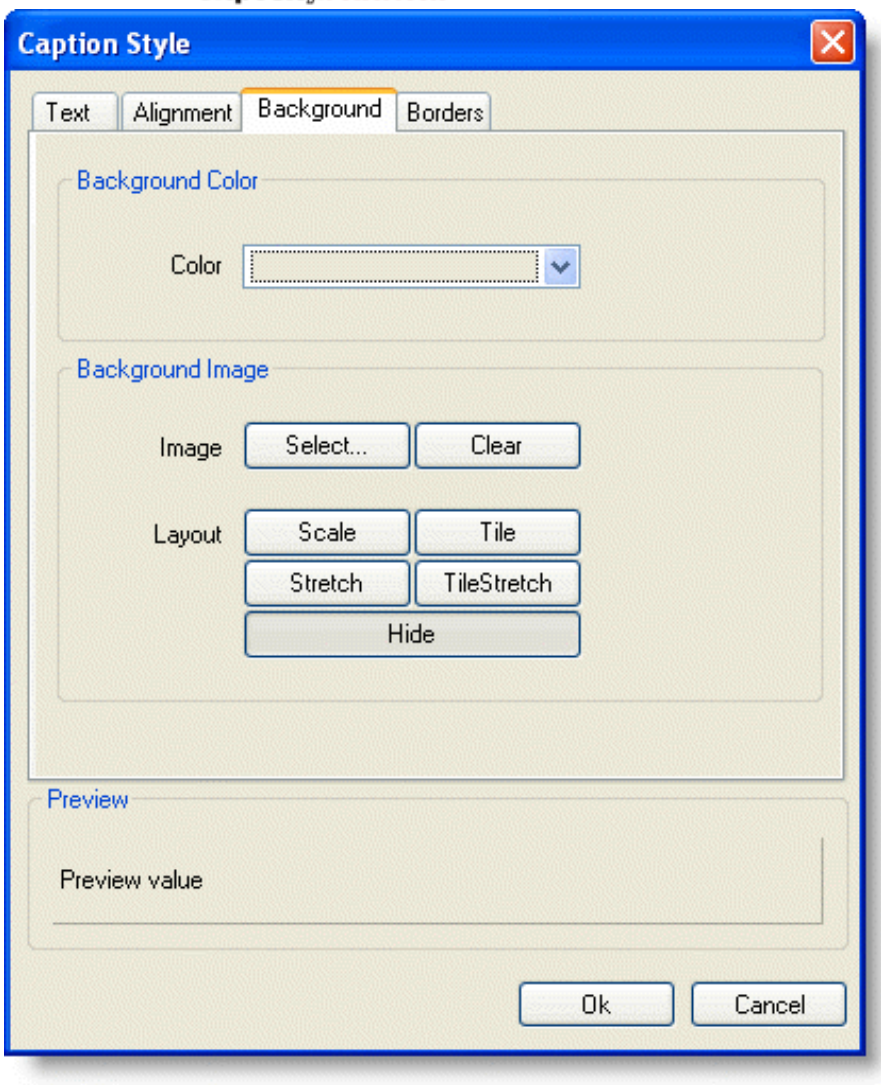
- **颜色**：为一个单元格选择一种背景颜色。

在**背景图片**区域可以用到下列选项：

- **图像**：单击选择按钮来选择一个图像或单击清除按钮来删除一个图像。
- 布局：可用于在 Scale, Tile, Stretch, TileStretch 或隐藏之间进行切换。

边框

边框标签可以用于设置边框与页边距。



在**边框**区域可以用到下列选项：

- **样式**：包括以下选项，“无”、“平面”、“重叠”、“凸起”、“嵌入”、“凹槽”，“袋状”或“点状”。有关边框样式不同选项的详细信息，请参阅 C1.Win.C1FlexGrid.BorderStyleEnum Enumeration。
- **宽度**：增加或减少该值来扩大或缩小边框的宽度。
- **颜色**：给边框选择一个颜色。
- **方向**：包括如下选项，水平垂直均有、水平或垂直。有关边框方向不同选项的详细信息，请参阅 BorderDirEnum Enumeration。

在**页边距**区域可以用到下列选项：

- **左边**：增加或减少该值来扩大或缩小左边的页边距。
- **顶端**：增加或减少该值来扩大或缩小顶端的页边距。
- **右边**：增加或减少该值来扩大或缩小右边的页边距。
- **底部**：增加或减少该值来扩大或缩小底部的页边距。

2.2 C1FlexGrid 智能标签

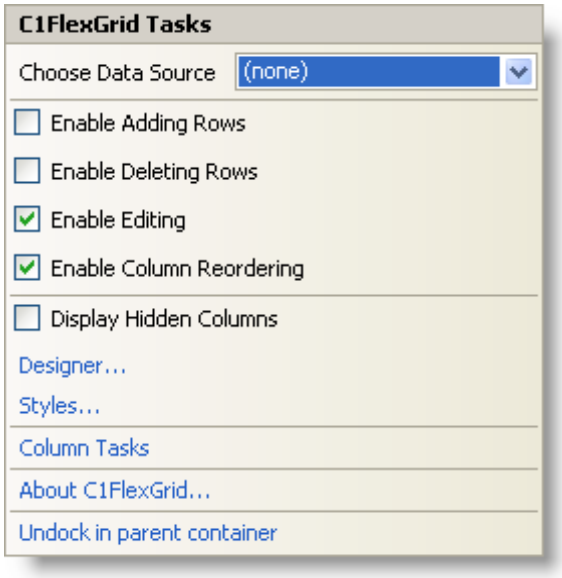
智能标签（ ）代表着捷径**任务**菜单，可以提供控件中最常用的几种属性。

C1FlexGrid 中通过智能标签提供了两个任务菜单，C1FlexGrid 任务菜单（第 34 页）和列任务菜单（第 36 页）。

2.2.1 C1FlexGrid 任务菜单

在 **C1FlexGrid 任务**菜单中，你可以快速访问 **C1FlexGrid 列编辑器**和 **C1FlexGrid 样式编辑器**，以及设置以下属性：允许添加新建、允许删除、允许编辑和允许拖动。

要进入 **C1FlexGrid 任务**菜单，只需要单击表格右上角的智能标签（ ）。这步操作就可以打开 **C1FlexGrid 任务**菜单。



选择数据源

在“选择数据源”对话框中单击下拉箭头就会打开一个可用的数据源的列表，并且允许你自己添加一个新的数据源。要想将一个新的数据源添加到项目，需要单击“**添加项目数据源**”来打开“数据源配置向导”。

有关如何将一个新的数据源添加到项目的详细信息，请参阅“绑定到数据源”（第 95 页）。

启用添加行

选择“启用添加行”复选框，将“允许添加新建”的属性设置为“真”，并且允许添加新的行到表格。默认设为“未被选中”。

启用删除行

选择“启用删除行”复选框，将“允许删除”的属性设置为“真”，并且

允许从表格中删除行。默认设为“未被选中”。

启用编辑

选择“启用编辑”复选框，将“允许编辑”的属性设置为“真”，并且允许在表格中进行编辑。默认设为“被选中”。

启用列重新排序

选择“启用列重新排序”复选框，将“允许拖动”的属性设置为“列”，并且允许在表格中对列进行拖动。默认设为“被选中”。

显示隐藏的列

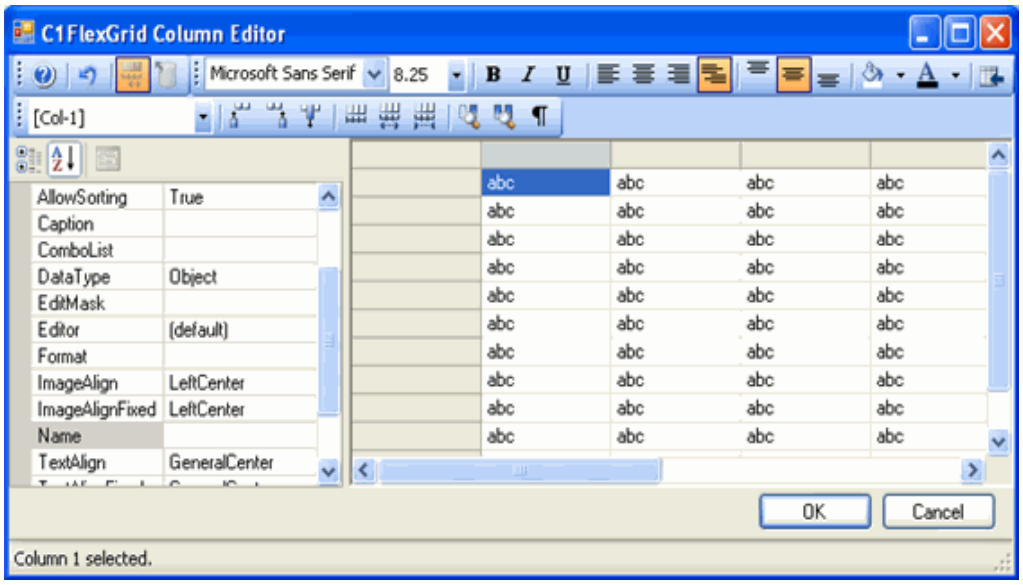
选择“显示隐藏的列”复选框，将隐藏列的“可见性”属性设置为“真”，并显示它们。默认设为“未被选中”。

设计器

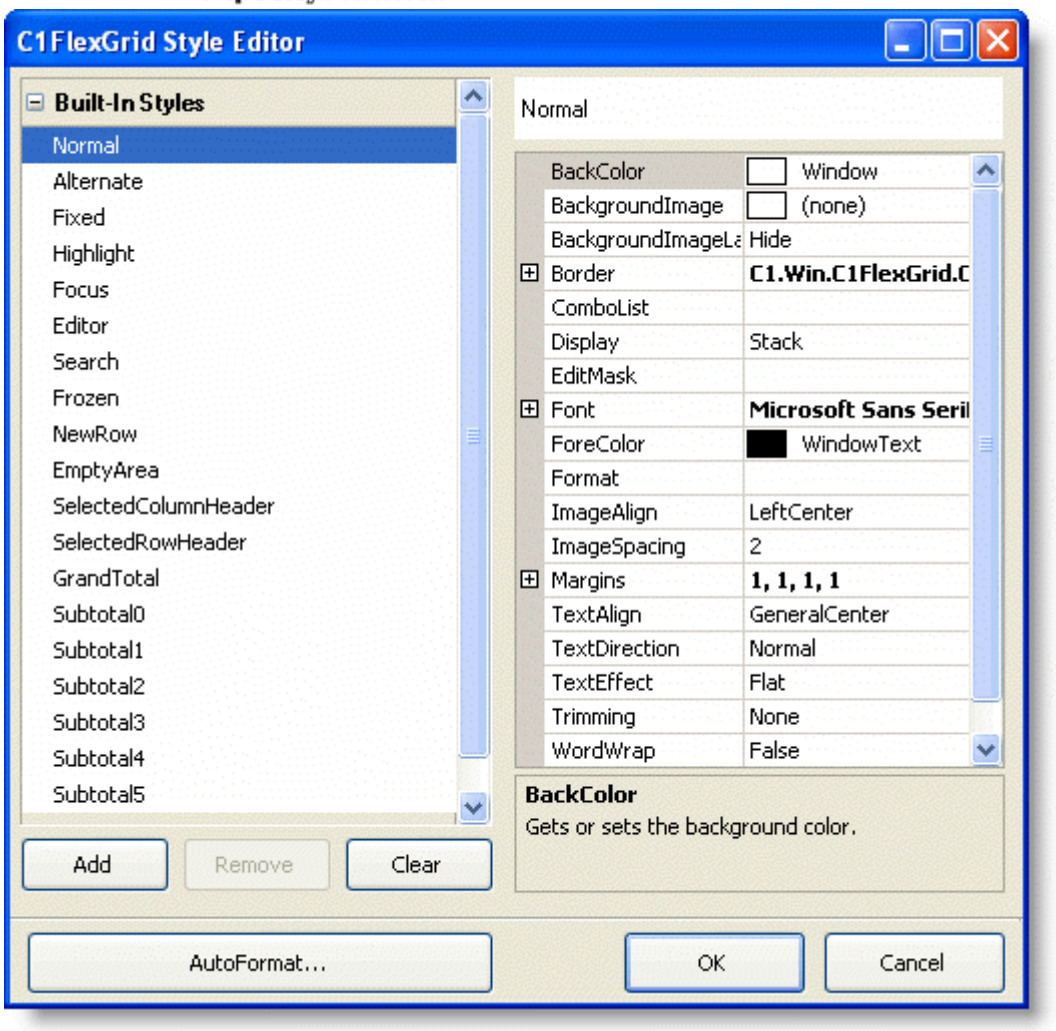
单击**设计器**来打开 C1FlexGrid 列编辑器。

有关如何使用 C1FlexGrid 列编辑器来对列进行编辑的详细信息，请参阅“C1FlexGrid 列编辑器”（第 25 页）。

样式



单击**样式**来打开 C1FlexGrid 样式编辑器。



有关如何使用 C1FlexGrid 样式编辑器来自定义单元格的外观的更多详细信息，请参阅“C1FlexGrid 样式编辑器”（第 27 页）。

列任务

单击“列任务”来打开**列任务**菜单。有关**列任务**菜单的更多详细信息，请参阅“列任务菜单”（第 36 页）。

关于 C1FlexGrid

点击关于 C1FlexGrid 来显示“关于 ComponentOne C1FlexGrid”对话框，这对找到 C1FlexGrid 的版本号是有帮助的。

停靠在父容器/取消停靠在父容器

点击**停靠在父容器**来将 C1FlexGrid 的“**停靠**”属性设置为“**填充**”。

如果 C1FlexGrid 已经停靠在父容器，取消 C1FlexGrid 停靠在父容器这个选项是可用的。单击**取消停靠在父容器**来将 C1FlexGrid 的“**停靠**”属性设置为“**无**”。

2.2.2 列任务菜单

“**列任务**”菜单允许你对一个列的列标题、数据字段、数据类型、编辑掩码、格式字符串和组合列表进行设置，并可以设置以下属性：允许排序、允许编辑、允许调整、允许拖动、允许合并以及可见性。

要访问“**列任务**”菜单，可以通过一下两种方式，或点击表格中的列，或从 **C1FlexGrid 任务**菜单中选择“**列任务**”。

列标题

在“**列标题**”对话框中输入一个标题来对这一列的标题属性进行设置。

数据字段

在“**数据字段**”对话框中单击下拉箭头来打开一个数据源中的可用字段的列表。

数据类型

在“数据类型”对话框中单击下拉箭头来打开一个的可用数据类型的列表。

编辑掩码

在“**编辑掩码”**对话框中单击省略号按钮来打开“输入掩码”对话框。

格式字符串

在“**格式字符串**”对话框中单击省略号按钮来打开“格式字符串”对话框。

组合列表

在“**组合列表**”对话框中单击省略号按钮来打开“**组合列表**”对话框。

允许排序

选择“**允许排序**”复选框，将“**允许排序**”属性设置为“真”，并允许在该列进行排序。默认设为“被选中”。

允许编辑

选择“**允许编辑**”复选框，将“**允许编辑**”属性设置为“真”，并允许在该列进行编辑。默认设为“被选中”。

允许调整

选择“**允许调整**”复选框，将“**允许调整**”属性设置为“真”，并允许在该列进行调整。默认设为“被选中”。

允许拖动

选择“**允许拖动**”复选框，将“**允许拖动**”属性设置为“真”，并允许在该列进行拖动。默认设为“被选中”。

允许合并

选择“**允许合并**”复选框，将“**允许合并**”属性设置为“真”，并允许在该列进行合并。默认设为“被选中”。

可见性

选择“**可见性**”复选框，将“**可见性**”属性设置为“真”，并允许该列在表格中可见。默认设为“被选中”。

标题样式

单击“**标题样式**”来打开所选列的**标题样式**编辑器，它允许你指定标题文本的属性，包括对齐，背景和边框。

列样式

单击“**列样式**”来打开所选列的**列样式**编辑器，它允许你指定该列的属性，包括对齐，背景和边框。

C1FlexGrid 任务

单击 **C1FlexGrid 任务**，你可以返回到 **C1FlexGrid 任务**菜单。有关 **C1FlexGrid 任务菜单**的详细信息，请参阅“C1FlexGrid 任务菜单”。

停靠在父容器/取消停靠在父容器

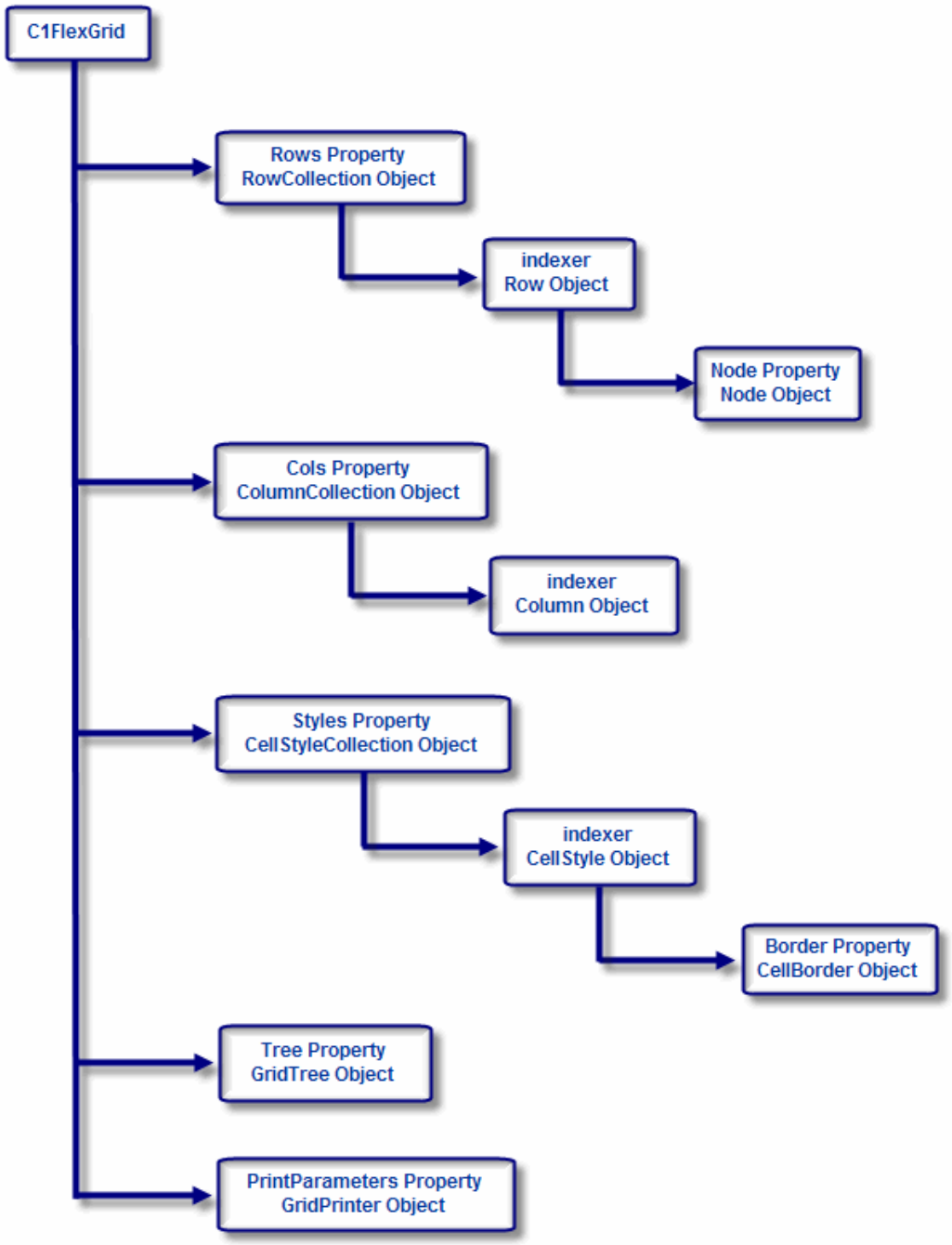
点击**停靠在父容器**来将 C1FlexGrid 的“**停靠**”属性设置为“**填充**”。

如果 C1FlexGrid 已经停靠在父容器，取消 C1FlexGrid 停靠在父容器这个选项是可用的。单击**取消停靠在父容器**来将 C1FlexGrid 的“**停靠**”属性设置为“**无**”。

3. 使用 C1FlexGrid 控件

你可以使用 C1FlexGrid 控件来对一个表格格式中的数据进行显示，编辑，组和总结。该表格可以绑定到一个数据源，它可以对自己的数据进行管理。

C1FlexGrid 控件有一个包含以下元素的丰富的对象模型：



以下的几个标题将引导你熟悉 C1FlexGrid 控件的几个主要特点：

行与列（第 41 页）

介绍了应该如何设置表格尺寸和布局。

单元格选择（第 43 页）

介绍了“当前单元格”和“选择”的概念。

单元格区域（第 45 页）

介绍了应该如何将一个单元格的集合作为一个单元来操作。

单元格图像（第 45 页）

介绍了应该如何在一个单元格中显示图像。

设置单元格格式（第 46 页）

介绍了应该如何通过设置数字、日期和布尔值的格式，或为一个单个的单元格或一个单元格的集合改变字体、颜色、对齐方式来自定义表格的外观。

编辑单元格（第 51 页）

介绍了应该如何实现简单的文字编辑，下拉列表和组合列表，单元格按钮，编辑掩码和数据验证。

合并单元格（第 62 页）

介绍了应该如何通过建立“分类”意见来突出数据的关系并更改表格的显示，以便使具有类似内容的单元格合并到一起。

概述和汇总数据（第 67 页）

介绍了应该如何将分类汇总添加到表格和如何建立一个树形大纲。

保存，加载，并打印（第 84 页）

介绍了你应该如何对一个表格的内容或格式进行保存，并在稍后重新加载，或利用其他应用程序，例如 Microsoft Access 和 Excel，来互换表格的数据。本节还介绍了你应该如何对表格进行打印。

C1FlexGrid 属性组

介绍了一个按功能交叉引用的 C1FlexGrid 主要属性的图。

3.1 行和列

一个 C1FlexGrid 控件包括由行和列。行和列的集合受到行和列属性的影响。

当把表格绑定到一个数据源，行和列的数目取决于可用的数据源中的数据有多少。在绑定模式下，你可以使用集合的 Count 属性将它们设置为任意值。例如，下面的代码将表格的尺寸设置为 500 行乘以 10 列：

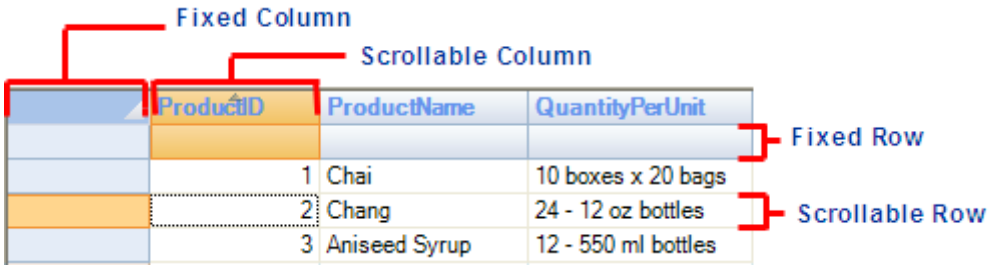
- Visual Basic

```
_flex.Rows.Count = 500
_flex.Cols.Count = 10
```

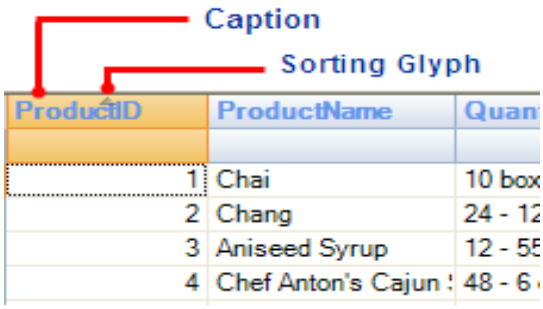
- C#

```
flex.Rows.Count = 500;
_flex.Cols.Count = 10;
```

行和列有两个基本类型：固定的和滚动的。（通过 Count 属性被还原的数值包括了固定的和滚动的单元格）。当用户垂直滚动表格时，固定行停留在表格顶部；而当用户水平滚动表格时，固定列停留在表格左侧。



固定的单元格用于显示行和列的标题信息。



你可以通过使用行和列的集合中“**固定的**”属性来设置固定的行和列的数目。例如，下面的代码可以创建一个具有两个固定行和无固定列的表格：

- Visual Basic

```
flex.Rows.Fixed = 1
_flex.Cols.Fixed = 0
```

- C#

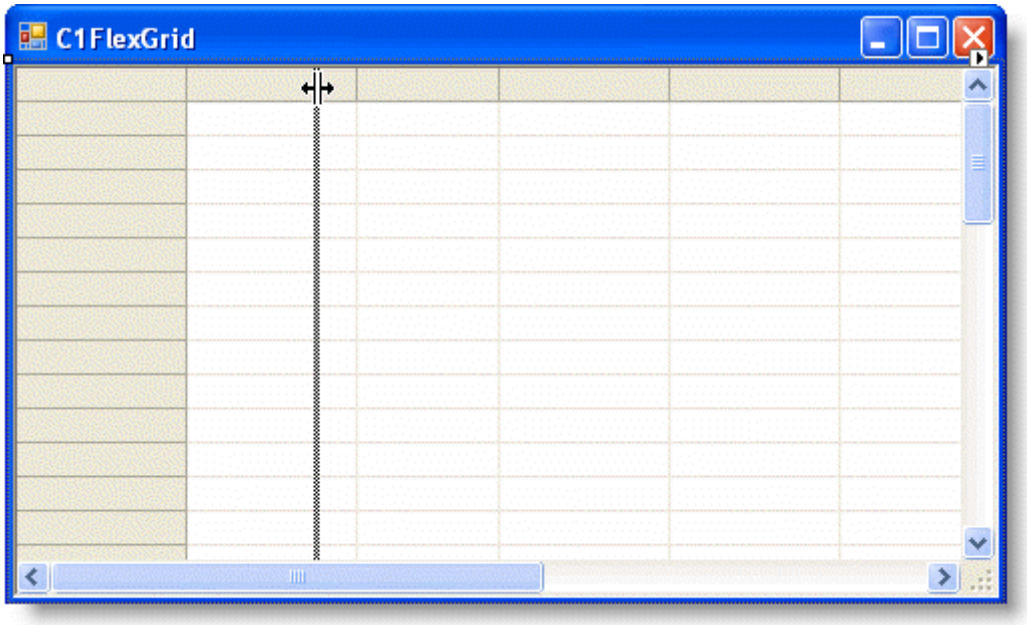
```
flex.Rows.Fixed = 1;
_flex.Cols.Fixed = 0;
```

行和列的集合也包含了在表格中插入、删除和移动行和列的方法。你可以使用“条目”属性（索引）来访问在每个集合中的独立的各个元素（行和列）。

只要你愿意，你就可以在设计时通过使用 **C1FlexGrid 列编辑器**来设置好表格列，而不是靠编写代码来执行这一步操作。有关如何使用 **C1FlexGrid 列编辑器**来对列进行编辑的更多详细信息，请参阅 C1FlexGrid 列编辑器。

3.1.1 列宽度

列的宽度取决于它的“宽度”属性。在设计时，可以在表格中直接设置“宽度”属性，或通过 **C1FlexGrid 列编辑器**来进行。在表格中，单击并拖动出现在列标题的右边缘的水平的双箭头可以重新调整列的尺寸。



垂直的虚线显示了表格将会如何被重新调整尺寸。向左拖动指针使列宽变小；将其向右边拖动使列宽变大。当调整的操作完成，列的“宽度”属性将被调整。

在 **C1FlexGrid 列编辑器**中或代码中，为列指定“宽度”属性值。有关 **C1FlexGrid 列编辑器**的更多详细信息，请参阅“C1FlexGrid 列编辑器”（第 25 页）。下面的代码可以设置第一列到第十列的 “宽度”的属性：

- Visual Basic

```
    _flex.Cols(1).Width = 10
```

- C#

```
    _flex.Cols[1].Width = 10;
```

为了防止一个特定的列被调整，请在无论是**列任务**菜单上，还是**列编辑器**或代码中，都将这一列的“允许调整”属性设置为“假”。有关**列任务**菜单的更多详细信息，请参阅“列任务菜单”（第 36 页）。下面的代码可以将第一列的“允许调整”属性设置为“假”：

- Visual Basic

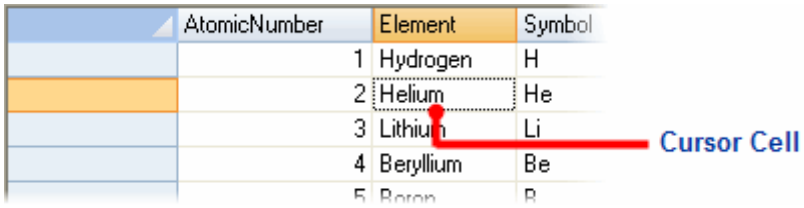
```
_flex.Cols(1).AllowResizing = False
```

- C#

```
_flex.Cols[1].AllowResizing = false;
```

3.2 单元格选择

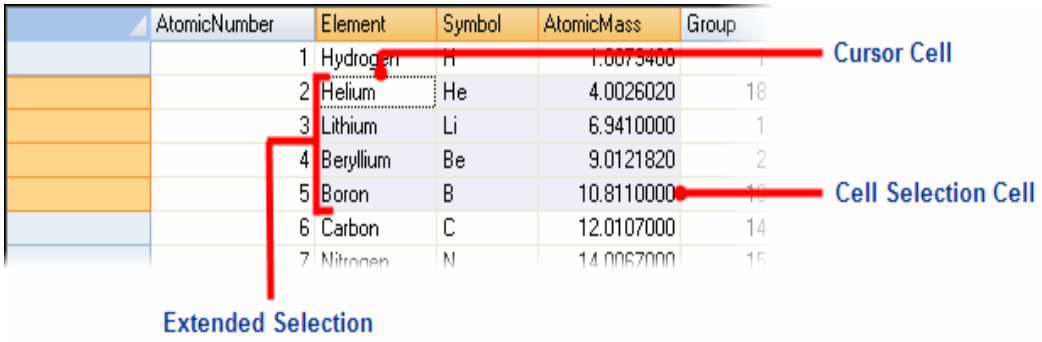
表格中有一个光标单元格，当表格时是激活的，它会显示一个焦点矩形。当表格是可编辑的，用户可以使用键盘或鼠标来移动这个光标并且编辑单元格的内容。



请注意，Office Visual Styles 也会通过高亮显示光标单元格所在的行和列标题，来显明光标单元格的位置。要想了解更多有关设置“视觉样式”的详细信息，请参阅“自定义外观，使用视觉样式”（第 165 页）。

你可以在代码中通过使用“行与列”属性来获取或设置当前单元格。例如，将任一属性设置为-1，可以隐藏光标。

表格支持**扩展选择**，单元格的矩形范围由两个对立的角落决定，即光标所在单元格和单元格选择的单元格。



请注意，Office Visual Styles 也通过高亮显示选定的单元格所在的行和列的标题来显明了扩展选择的位置。要想了解更多有关设置“视觉样式”的详细信息，请参阅“自定义外观，使用视觉样式”（第 165 页）。

你可以在代码中通过使用“行选择”和“列选择”属性来获取或设置单元格选择，或通过使用“选择”的方法也可以。

注意：当光标单元格变化了，该“选择”会自动复位。要在代码中创建扩展选项，无论是通过在“行选择”和“列选择”之前先设置“行与列”或直接使用“选择”的方法都可以。

“选择”的外观是由以下属性决定的：

定焦矩形决定了焦点矩形的类型是为了表明光标单元格而绘制的。

高亮显示决定了什么时候该“选择”应该被突出显示（通常是，当该控件有一个突出的重点时，或者从不）。

高亮和定焦都是决定了“选择”的外观的单元格样式（字体，颜色和边框）。

可供选择的类型是由“选择模式”的属性决定的。在默认情况下，表格支持常规的范围选择。你可以修改此行为来防止扩展选择，或者来按行、按列，或在列表框模式下进行选择（“列表框模式”可以允许你来选择个别行）。

当使用列表框选择模式时，你可以通过使用“选择”属性来获取或设置个别行的选择状态。你还可以通过使用“选择”属性来恢复选定的行的集合。例如，下面的代码可以选择符合一个条件的所有行：

- Visual Basic

```
' 在“销售”列中选择多于 8000 名销售的所有行。
_flex.SelectionMode = C1.Win.C1FlexGrid.SelectionModeEnum.ListBox
Dim index As Integer
For index = _flex.Rows.Fixed To _flex.Rows.Count - 1
If Val(_flex(index, "Sales")) > 80000 Then
_flex.Rows(index).Selected = True
End If
Next
Console.WriteLine("There are now {0} rows selected",
_flex.Rows.Selected.Count)
```

- C#

```
// 在“销售”列中选择多于 8000 名销售的所有行。
_flex.SelectionMode = SelectionModeEnum.ListBox;
for (int index = _flex.Rows.Fixed ; index < _flex.Rows.Count; index++)
{
if
(Microsoft.VisualBasic.Conversion.Val(System.Runtime.CompilerServices.Runti
meHelpers.GetObjectValue(_flex[index, "Sales"])) > 80000)
{
_flex.Rows[index].Selected = true;
}
}
Console.WriteLine("There are now {0} rows selected",
_flex.Rows.Selected.Count);
```

3.3 单元格区域

“单元格区域”对象可以使你将单元格的任意组合作为一个简单的单元来工作。例如，下面的代码创建了一个“单元格区域”的对象，清除了范围内的数据，并赋予了它一个自定义样式：

- Visual Basic

```
Dim rg As CellRange = _flex.GetCellRange(3, 3, 10, 10)
rg.Data = Nothing
rg.Style = _flex.Styles("MyRangeStyle")
```

- C#

```
CellRange rg = _flex.GetCellRange(3, 3, 10, 10);
rg.Data = null;
rg.Style = _flex.Styles["MyRangeStyle"];
```

“单元格区域”的对象具有可以恢复区域样式的 StyleNew 的属性，如果存在一个，或创建了一个新的，将其分配给该区域并返回它。在你不需要对格式进行全面控制的情况下，此属性是很方便的。例如，如果你想要做的是给这个区域设定一个红色的背景，你可以编写如下代码来实现：

- Visual Basic

```
Dim rg As CellRange = _flex.GetCellRange(3, 3, 10, 10)
rg.StyleNew.BackColor = Color.Red
```

- C#

```
CellRange rg = _flex.GetCellRange(3, 3, 10, 10);
rg.StyleNew.BackColor = Color.Red;
```

3.4 单元格图像

每个表格内的单元格除了在单元格中存储数据以外还都可以显示图像。这可以通过两种方式来实现：

你可以通过使用“设置单元格图像”的方法，来将一个“图像”对象分配到一个单元格。这种方法可以允许你指定任意图像到每个单元格，并且当图像与单元格中的数据没有什么关系时这是很有用的。例如，你可能想使用图片作为一项指标来表明单元格中的数据是无效的。

你可以将一个“图像映射”指定到一列，表格就会将单元格中的数据自动映射到一个相应的图像。当图像包含了代表性的数据的情况下，这种方法是很有用的。例如，图像可能含有表示产品类型的图标。

3.5 设置单元格格式

C1FlexGrid 控件的主要优势之一就是，具有自定义整个表格和每个个体的单元格外观的几乎任一方面的能力。

3.5.1 单元格的内容

为了控制单元格的内容如何被格式化，请将“格式化”属性设置到一个类似于你在.NET framework 中通过 String.Format 方法使用的格式字符串。例如，下面的代码显示了第一列的短日期和第二列的货币值：

- Visual Basic

```
' 短日期。
_flex.Cols(1).Format = "d"
' 货币值。
_flex.Cols(2).Format = "c"
```

- C#

```
// 短日期。
_flex.Cols[1].Format = "d";
// 货币值。
_flex.Cols[2].Format = "c";
```

单元格内容的格式也是可以通过使用**格式字符串**对话框在设计时就进行设置的。**格式字符串**对话框的访问路径有两种，一种是可通过**列任务**菜单。另一种是可通过 **C1FlexGrid 列编辑器**。

在“列任务”菜单上，单击“格式字符串”中的省略号按钮。

在 C1FlexGrid 列编辑器中，在左窗格中找到“格式”属性，并单击它旁边的省略号按钮。

注意：**格式字符串**对话框是特定列，而且只会更改所选定的列的“**格式**”属性。

你还是可以使用自定义格式，就像你在 Visual Basic “**格式**”功能中所使用的一样（例如，“＃，＃＃＃”，等等）。

检索单元格数据

你可以使用索引或“获取数据”方法来检索原始的表格数据。要检索格式过的数据，请改为使用 GetDataDisplay 的方法。例如：

- Visual Basic

```
' 短日期。
_flex.Cols(1).Format = "d"
' 货币值。
_flex.Cols(2).Format = "c"
_flex(1, 2) = 10000
Console.WriteLine("Raw value: {0}", _flex(1, 2))
Console.WriteLine("Display value: {0}", _flex.GetDataDisplay(1, 2))
' 行值：10000
```

- C#

```
// 短日期。
_flex.Cols[1].Format = "d";
// 货币值。
_flex.Cols[2].Format = "c";
_flex[1, 2] = 10000;
Console.WriteLine("Raw value: {0}", _flex[1, 2]);
Console.WriteLine("Display value: {0}", _flex.GetDataDisplay(1, 2));
// 行值：10000
```

3.5.2 单元格的外观

单元格的外观（对齐方式、字体、颜色、边框，等）是由“单元格样式”对象处理的。表格有一个“样式”属性，它包含了用于设置单元格格式的所有

样式的集合。这个集合有一些可以定义表格元素的外观的内置成员，如固定的和滚动的单元格，单元格选择，单元格定焦，等等。你可以更改这些样式来修改表格的外观，你也可以创建自己的自定义样式，并将它们分配给单元格，行或列。

改变内置的样式是改变表格外观的最简单的方式。例如，下面的代码可以将所选定的部分的背景显示为红色，字体为绿色粗体：

- Visual Basic

```
Dim cs As CellStyle = _flex.Styles.Highlight
cs.Font = New Font(_flex.Font, FontStyle.Bold)
cs.ForeColor = Color.Green
cs.BackColor = Color.Red
```

- C#

```
CellStyle  cs = _flex.Styles.Highlight;
cs.Font = new Font(_flex.Font,  FontStyle.Bold);
cs.ForeColor = Color.Green;
cs.BackColor = Color.Red;
```

你还可以创建你自己的样式，并将它们分配到单元格，行和列。例如，下面的代码可以创建一个自定义单元格样式并敬爱那个它分配给每一个第五行：

- Visual Basic

```
Dim cs As CellStyle = _flex.Styles.Add("Fifth")
cs.BackColor = Color.Gray
Dim idex%
For idex = _flex.Rows.Fixed To _flex.Rows.Count - 1 Step 5
_flex.Rows(idex).Style = cs
Next
```

- C#

```
CellStyle cs = _flex.Styles.Add("Fifth");
cs.BackColor = Color.Gray;
for (int index = _flex.Rows.Fixed ; index <= _flex.Rows.Count - 1; index +=
5)
{
_flex.Rows[index].Style = cs;
```

下面是一个例子，显示了如何创建自定义样式，并将它们分配到行，列和

单元格区域：

- Visual Basic

```
' 创建一个新的自定义样式。
Dim s As CellStyle = _flex.Styles.Add("MyStyle")
s.BackColor = Color.Red
s.ForeColor = Color.White
'  将新的样式分配到一列。
_flex.Cols(3).Style = _flex.Styles("MyStyle")
'  将新的样式分配到一行。
_flex.Rows(3).Style = _flex.Styles("MyStyle")
'  将新的样式分配到一个单元格区域。
Dim rg As CellRange = _flex.GetCellRange(4, 4, 6, 6)
rg.Style = _flex.Styles("MyStyle")
```

- C#

```
// 创建一个新的自定义样式。
CellStyle  s = _flex.Styles.Add("MyStyle");
s.BackColor = Color.Red;
s.ForeColor = Color.White;
// 将新的样式分配到一列。
_flex.Cols[3].Style = _flex.Styles["MyStyle"];
// 将新的样式分配到一行。
_flex.Rows[3].Style = _flex.Styles["MyStyle"];
// 将新的样式分配到一个单元格区域。
CellRange rg = _flex.GetCellRange(4,4,6,6);
rg.Style = _flex.Styles["MyStyle"];
```

只要你愿意，你就可以在设计时使用 **C1FlexGrid 样式编辑器**来设置好样式，而不是靠编写代码来执行这一步操作。有关如何使用 **C1FlexGrid 样式编辑器**来自定义单元格的外观的更多详细信息，请参阅“C1FlexGrid 样式编辑器”（第 27 页）。

3.5.3 有条件地设置格式

要根据单元格的内容设置其格式，你可以基于其内容使用“单元格变更”事件来为单元格选择一种样式。例如，下面的代码可以根据单元格的内容为“大货币值”创建一个新的样式，并将其应用于单元格：

- Visual Basic

```
Dim cs As C1.Win.C1FlexGrid.CellStyle
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 为“大货币值”创建一个自定义样式。
cs = _flex.Styles.Add("LargeValue")
cs.Font = New Font(Font, FontStyle.Italic)
cs.BackColor = Color.Gold
End Sub
' 根据单元格的内容来设置其格式。
Private Sub _flex_CellChanged(ByVal sender As Object, ByVal e As
RowColEventArgs) Handles _flex.CellChanged
' 将货币值大于 50,000 的标记为“大货币值”（通过将“样式”设置为
“无”来重新设置其他的）。
Dim cs As CellStyle
If _flex(e.Row, e.Col).ToString >= 50000 Then
cs = _flex.Styles("LargeValue")
_flex.SetCellStyle(e.Row, e.Col, cs)
End If
```

- C#

```
CellStyle cs;
private void Form1_Load(object sender, EventArgs e)
{
// 为大货币值创建一个自定义样式。
cs = _flex.Styles.Add("LargeValue");
cs.Font = new Font(Font, FontStyle.Italic);
cs.BackColor = Color.Gold;
}
// 根据单元格的内容来设置其格式。
private void _flex_CellChanged( object sender, RowColEventArgs e)
{
// 将货币值大于 50,000 的标记为“大货币值”（通过将“样式”设置为
“无”来重新设置其他的）。 if
(Microsoft.VisualBasic.CompilerServices.Conversions.ToDouble(_flex[e.Row,
e.Col].ToString()) >= 50000)
{
cs = _flex.Styles["LargeValue"];
_flex.SetCellStyle(e.Row, e.Col, cs);
}
```

3.5.4 自绘单元格

即使 “单元格样式”对象已经通过单元格外观（背景色和前景色、对齐方式、字体、边距、边框，等等）提供了很多控件，有时这仍是不够的。你可能想使用渐变背景，或直接在单元格中画一些自定义的图形。在这些情况下，你可以使用“绘制模式”属性和“自绘单元格”事件，通过每一个单元格是如何绘制的来实现全面控制。

“绘制模式”属性决定了 “自绘单元格”事件是否被激发。该事件可以是你兼顾到每一个单元格的视觉效果。 “自绘单元格”事件中的参数，允许你对所显示的数据以及用于显示数据的样式，做出更改，甚至允许你完全接管并绘制单元格中你所想要的任何东西。

你可以通过在事件处理程序中设置 e.Text 和 e.Image 参数，来改变即将显示在单元格中的文本和图像。你也可以通过设置 e.Style 属性来改变将用于显示单元格的样式。请注意，你不应该修改样式参数的属性，因为这可能会影响到其他单元格。相反，你要分配一个新的“**单元格样式**”对象到“样式”参数。例如，不要设置 **e.Style.ForeColor= Color.Red**，而应该通过使用 **e.Style = _flex.Styles["RedStyle"]**将一个全新的样式分配给参数。

你还可以使用自己的绘图代码在单元格中进行绘制，甚至用命令将你的自定义代码与 **e.DrawCell** 方法合并到一起。例如，你可以使用 GDI 命令来绘制单元格的背景，然后调用 **e.DrawCell** 来显示单元格边框和内容。

可用的示例项目

有关高级的**自绘单元格**的一个范例，请参阅“ComponentOne 帮助中心”的 **RtfGrid** 样品。

下面的代码显示了一个例子。它使用了渐变画笔来绘制所选定的单元格的背景。首先，代码会设置“绘制模式”属性，显示一个“**线性渐变画笔**”的对象并且在表格重新调整大小时更新画笔：

- Visual Basic

```
Dim m_GradientBrush As System.Drawing.Drawing2D.LinearGradientBrush
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 通过自绘单元格来使用画笔。
m_GradientBrush = New
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.SteelBlue, Color.White, 45)
' 使用自绘来添加渐变。
_flex.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
End Sub
Private Sub _flex_Resize(ByVal sender As Object, ByVal e As
System.EventArgs) Handles _flex.Resize
' 当控件调整大小时更新渐变画笔。
m_GradientBrush = New
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.SteelBlue, Color.White, 45)
End Sub
```

- C#

```
System.Drawing.Drawing2D.LinearGradientBrush m_GradientBrush;
private void Form1_Load(object sender, EventArgs e)
{
// 通过自绘单元格来使用画笔。
m_GradientBrush = new
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.SteelBlue, Color.White, 45);
// 使用自绘来添加渐变。
_flex.DrawMode = DrawModeEnum.OwnerDraw;
}
private void _flex_Resize( object sender, System.EventArgs e)
{
// 当控件调整大小时更新渐变画笔。
m_GradientBrush = new
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.SteelBlue, Color.White, 45);
}
```

第二步是处理“自绘单元格”事件，并使用自定义画刷来绘制单元格的背景。在这个例子中，通过在事件参数中使用 “绘制单元格”方法，表格自己处理了自己的前景：

- Visual Basic

```
Private Sub _flex_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles _flex.OwnerDrawCell
' 使用渐变画笔来绘制所选定的单元格的背景。
If _flex.Selection.Contains(e.Row, e.Col) Then
' 绘制背景。
e.Graphics.FillRectangle(m_GradientBrush, e.Bounds)
' 让表格绘制其内容。
e.DrawCell(C1.Win.C1FlexGrid.DrawCellFlags.Content)
' 我们已完成这个单元格的绘制。
e.Handled = True
End If
```

- C#

```
    private void _flex_OwnerDrawCell( object sender, OwnerDrawCellEventArgs
e)
    {
    // 使用渐变画笔来绘制所选定的单元格的背景。
    if (_flex.Selection.Contains(e.Row, e.Col))
    {
    // 绘制背景。
    e.Graphics.FillRectangle(m_GradientBrush, e.Bounds);
    // 让表格绘制其内容。
    e.DrawCell(DrawCellFlags.Content);
    // 我们已完成这个单元格的绘制。
    e.Handled = true;
    }
```

3.6 编辑单元格

默认情况下，C1FlexGrid 控件允许用户通过在单元格中打字来编辑单元格。你可以通过“允许编辑”属性设置为“假”来阻止用户对表格进行编辑。你还可以通过将“允许编辑”属性设“假”来阻止用户对特定列进行编辑。（当表格被绑定到一个数据源上，它会检测哪些列是可编辑的并自动设置“允许编辑”属性。）

要开始编辑一个单元格，用户可以：

- 开始在单元格中输入。这将替换单元格原有的内容。
- 按 F2 键或 ENTER 键。这使得表格处于编辑模式下，并将当前单元格的内容放到编辑器。
- 双击一个单元格。这与按 F2 键有相同的效果，但光标会出现在被点击的单元格。

基本的编辑模式允许用户在单元格中输入值。如果正在被编辑的列有一个特定的数据类型，由用户输入的值会自动转换成适当的数据类型。如果用户输入了不能转换成适当的数据类型的值，表格会激活“表格错误”事件并忽略该编辑。

对于许多应用来说，基本的编辑就足够了，但 C1FlexGrid 含有这样的属性和事件，它能让你控制编辑过程，并提供选择清单，编辑按钮和高级的验证控制。从版本 2.5 开始，C1FlexGrid 也有了外部编辑器的内置支持。这使你可以将任何一个控件作为表格编辑器使用（例如，现在你可以将 C1Input 控件作为表格编辑器使用。）

在以下主题中主要介绍了这些功能。

3.6.1 列表和组合

在许多应用中，单元格都有一个关于可能值的定义明确的列表。在这些情况下，你可以让用户从**下拉列表**中选择值。要做到这一点，必须建立一个字符串，其中包含所有的以管道字符分隔出来的选择（例如，“真|假|不知道”），并将它分配到“组合列表”属性。每一列都可能有不同的列表。设置“组合列表”属性，使表格在单元格旁边显示一个下拉框。用户可以单击此下拉框（或按 F2 键）来显示该单元格中可用选择的列表。

另一种常见的情况是即使单元格有共同价值的一个列表，但也应允许用户键入别的东西。这可以通过**下拉组合**，文本框和下拉列表的组合来实现。要创建一个组合，只需用管道字符（例如“|真|假不知道”）来启动选择列表，然后将它像以前一样分配到“组合列表”属性。

例如，下面的代码会使表格在第一列上显示包含了颜色名称的下拉组合列表，并在第二列上显示一个下拉组合。当对第一列进行编辑时，用户必须从列表中选择一个值。当对第二列进行编辑时，用户也要选择一个值或输入什么别的东西：

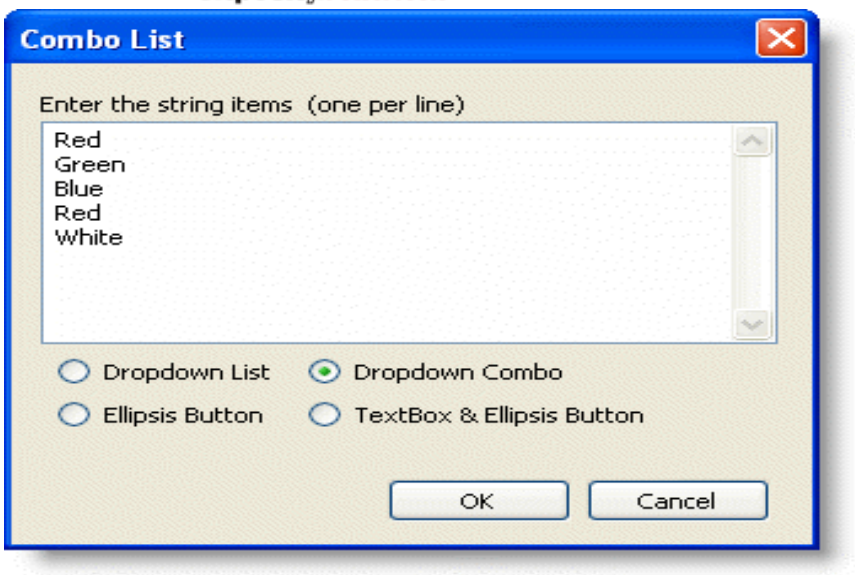
- Visual Basic

```
' 下拉列表。
_flex.Cols(1).ComboList = "Red|Green|Blue|Red|White"
' 下拉组合。
_flex.Cols(2).ComboList = "|Red|Green|Blue|Red|White"
```

- C#

```
// 下拉列表。
_flex.Cols[1].ComboList = "Red|Green|Blue|Red|White";
// 下拉组合。
_flex.Cols[2].ComboList = "|Red|Green|Blue|Red|White";
```

你也可以在设计时使用“**组合列表**”对话框来设置“组合列表”属性。“**组合列表**”对话框允许你选择你是想让列表显示为一个**下拉列表**，**下拉组合**，**省略号按钮**，还是**文本框**和**省略号按钮**。



你可以通过“**列任务**”菜单，或通过 **C1FlexGrid 列编辑器**来访问“组合列表”对话框。

- 在“**列任务**”菜单上，单击**组合列表**框中的**省略号**按钮。
- 在 **C1FlexGrid 列编辑器**中，在左窗格中找到的“组合列表”属性，并单击它旁边的**省略号**按钮。

注意：**组合列表**对话框是特定的列，并且只会更改所选定列的“组合列表”属性。

在某些情况下，在同一列中的单元格可能需要不同的列表。例如，一个属性列表可能会在第一列上显示显示所有的属性，并在第二列上显示他们的值。该值取决于属性本身，所以有效的选项一行接一行地改变。在这种情况下，你应该捕获“编辑前”事件，并且为当前单元格设置“组合列表”属性到一个适当的列表。“组合列表”属性适用于整个表格。

内置的“组合框”提供了一个默认的自动搜索功能。当用户输入一个值，选中的部分将移动到下一个适配的对象。您可以通过使用“编辑选项”属性来禁用此功能，也可以通过使用“自动搜索延迟”属性，在表格重新设置自动搜索缓冲之前来控制时间。

内置的“组合框”也有一个像 Visual Studio 的“属性”窗口中的编辑器那样的自动循环功能。当你双击一个具有与它相关联的列表的单元格，表格会自动选择下一个值。你也可以通过使用“编辑选项”属性来禁用此功能。

3.6.2 复选框

默认情况下，表格会将值以复选框的形式在布尔列显示（列的类型是由列对象的“数据类型”属性决定的）。如果你不想让让布尔值以复选框的形式显示出来，请将列的格式属性设置为一个字符串，此字符串包含会显示为“真”

和“假”的值。例如：

- Visual Basic

```
_flex.Cols("bools").Format = "Yes;No"
```

- C#

```
_flex.Cols["bools"].Format = "Yes;No";
```

在未绑定模式下，你可以通过使用 GetCellCheck 和 SetCellCheck 属性将复选框添加到任意单元格。复选框将会随着单元格中的任何文本来显示，而且你可以使用列的“图像对齐”属性来设置他们的位置。

复选框有两种类型：布尔和三态。布尔复选框可以在 CheckEnum.Checked 和 CheckEnum.Unchecked 状态之间进行切换。三态复选框可以在 CheckEnum.TSChecked， CheckEnum.TSUnchecked 和 CheckEnum.TSGrayed 之间进行循环切换。

如果某单元格有一个复选框并且“允许编辑”属性被设置为“真”，用户可以通过用鼠标点击或按空格键或回车键来改变复选框的状态。

默认情况下，用鼠标或键盘切换复选框的值会将所有选中的复选框都切换掉。你可以通过使用“编辑选项”属性来禁用此功能。

3.6.3 值映射列表

如上所述的“组合列表”属性可以确保从列表中选择单元格的值。由用户选择的值被转换成适合于列的类型并存储在表格中，确切地说，是如果用户输入了值的话。

在许多情况下，单元格可以从定义明确的列表中采用一个值，但你想要显示的是一个包含实际值的〉用户界面友好的版本。例如，如果一个列包含产品编码，你可能想存储它的代码，但显示的产品名称来代替。

这可以通过“数据映射”属性来实现。此属性包含一个 **IDictionary** 对象的引用，它在什么是存储在表格中的什么是用户可见的之间建立了一个映射（IDictionary 接口界面是在系统中定义的。集合的命名空间是由哈希表之类实现的）。

例如，下面的代码可以创建一个的包含颜色值和他们的名字的数据映射。颜色存储于表格中，而向用户显示的是他们的名字：

- Visual Basic

```
Dim dtMap As Hashtable = New Hashtable()
dtMap.Add(Color.Red, "Apple")
dtMap.Add(Color.Green,          "Forest")
dtMap.Add(Color.Blue,   "Sky")
dtMap.Add(Color.Black, "Coal")
dtMap.Add(Color.White,          "Snow")
_flex.Cols(1).DataType = GetType(Color)
_flex.Cols(1).DataMap = dtMap
```

- C#

```
System.Collections.Hashtable dtMap = new System.Collections.Hashtable();
dtMap.Add(Color.Red, "Apple");
dtMap.Add(Color.Green,          "Forest");
dtMap.Add(Color.Blue,   "Sky");
dtMap.Add(Color.Black, "Coal");
dtMap.Add(Color.White,          "Snow");
_flex.Cols[1].DataType = typeof(Color);
_flex.Cols[1].DataMap = dtMap;
```

配备了 **Idictionary** 接口的任何一类都可以作为一个“**数据映射**”来使用。例如，**Hashtable**，**ListDictionary** 和 **SortedList** 都提供了有效的数据映射。所不同的是，当他们在可编辑的列中使用时，下拉列表中条目的顺序将取决于类。

SortedList 类通过键值将各条目分类，**Hashtable** 通过任意顺序，而 **ListDictionary** 保持了那些条目被添加到列表中的顺序。正因为如此，**ListDictionary** 通常是“数据映射”的最佳选择。

请注意，当单元格正在编辑中时，数据映射的键值必须是同一类型的。例如，如果一个列包含短整数（Int16），那么，与此列相关联的任何数据映射也应该是短整型的键值。如果将正整数（INT32）作为键值将不起作用。

下面的例子显示了它们的区别：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 按 key 排序。
Dim sl As New SortedList()
sl.Add("0",         "Zero")
sl.Add("1",         "One")
sl.Add("2",         "Two")
sl.Add("3",         "Three")
' 保持添加顺序。
Dim ld As New Specialized.ListDictionary()
ld.Add(0, "Zero")
ld.Add(1, "One")
ld.Add(2, "Two")
ld.Add(3, "Three")
' 任意顺序。
Dim ht As New Hashtable()
ht.Add(0, "Zero")
ht.Add(1, "One")
ht.Add(2, "Two")
ht.Add(3, "Three")
_flex.Cols(1).DataMap = sl
_flex.Cols(1).Caption = "SortedList"
_flex.Cols(2).DataMap = ld
_flex.Cols(2).Caption = "ListDictionary"
_flex.Cols(3).DataMap = ht
_flex.Cols(3).Caption = "HashTable"
End Sub
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e);
{
// 按 key 排序。
System.Collections.SortedList sl = new System.Collections.SortedList();
sl.Add("0",         "Zero");
sl.Add("1",         "One");
sl.Add("2",         "Two");
sl.Add("3",         "Three");
// 保持添加顺序。
System.Collections.Specialized.ListDictionary ld = new
System.Collections.Specialized.ListDictionary();
ld.Add(0, "Zero");
ld.Add(1, "One");
ld.Add(2, "Two");
ld.Add(3, "Three");
// 任意顺序。
System.Collections.Hashtable ht = new System.Collections.Hashtable();
ht.Add(0, "Zero");
ht.Add(1, "One"); ht.Add(2,       "Two");
ht.Add(3, "Three");
_flex.Cols[1].DataMap = sl;
_flex.Cols[1].Caption = "SortedList";
_flex.Cols[2].DataMap = ld;
_flex.Cols[2].Caption = "ListDictionary";
_flex.Cols[3].DataMap = ht;
_flex.Cols[3].Caption = "HashTable";
}
```

此外，如果列的“数据类型”属性被设置为一个枚举，表格将自动建立并使用一个含枚举中每个值的名称的数据映射。例如，下面的代码可以创建一个包含国家的枚举。国家的值存储在表格中，向用户显示的是他们的名字：

- Visual Basic

```
Private Enum Countries
NewYork
Chicago
NewOrleans
London
Paris
End Enum
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
_flex.Cols(1).DataType = GetType(Countries)
End Sub
```

- C#

```
private enum Countries
{
NewYork,
Chicago,
NewOrleans,
London,
Paris
}
private void Form1_Load(object sender, EventArgs e)
{
_flex.Cols[1].DataType = typeof(Countries);
}
```

3.6.4 单元格按钮

某些类型的单元格，除了文本框或下拉列表以外，可能还需要更复杂的编辑器。例如，如果一个列包含文件名称或颜色，它应该可以用一个对话框来编辑。

在这些情况下，你应该将 “组合列表”属性设置为省略号（ “...” ）。该控件将在单元格旁边显示一个按钮，当用户点击它时，将触发“单元格按钮点击”事件。你可以捕获该事件，显示对话框，并使用用户的选择来更新单元格的内容。

如果你在省略号前添加管道字符，那么用户也将被允许通过在单元格输入的方式来编辑单元格的内容。

默认情况下，单元格的按钮显示省略号。您可以使用“单元格按钮图像”属性将图片指定到单元格按钮。

下面的示例演示了你应该如何使用单元格按钮来显示一个拾色器对话框，并在一列中选择一种颜色。

- Visual Basic

```
' 建立色列。
Dim c As C1.Win.C1FlexGrid.Column = _flex.Cols(1)
c.DataType = GetType(Color)
' 显示单元格按钮。
c.ComboList = "..."
```

- C#

```
// 建立色列。
Column c = _flex.Cols[1];
c.DataType = typeof(Color);
// 显示单元格按钮。
c.ComboList = "...";
```

通过此代码设置该列，用户可以通过点击一个按钮来从一个对话框中选择一个颜色。下一步处理单元格按钮点击的代码：

- Visual Basic

```
Private Sub _flex_CellButtonClick(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles _flex.CellButtonClick
' 创建“拾色器”对话框。
Dim clrDlg As New ColorDialog()
' 初始化对话框。
If _flex(e.Row, e.Col) Is GetType(Color) Then
clrDlg.Color = _flex(e.Row, e.Col)
End If
' 从对话框中获取新的颜色，并将它分配到单元格。
If clrDlg.ShowDialog() = Windows.Forms.DialogResult.OK Then
_flex(e.Row, e.Col) = clrDlg.Color
End If
End Sub
```

- C#

```
private void _flex_CellButtonClick( object sender, RowColEventArgs e)
{
// 创建“拾色器”对话框。
ColorDialog clrDlg = new ColorDialog();
// 初始化对话框。
if (_flex[e.Row, e.Col] == typeof(Color))
{
clrDlg.Color = (Color)_flex[e.Row, e.Col];
}
// 从对话框中获取新的颜色，并将它分配到单元格。
if (clrDlg.ShowDialog() == DialogResult.OK)
{
_flex[e.Row, e.Col] = clrDlg.Color;
}
}
```

3.6.5 掩码

C1FlexGrid 控件还支持屏蔽编辑。当用户输入时，这种类型的编辑可以使用一种输入掩码来提供一个模板，并自动验证该输入。该掩码由“编辑掩码”属性明确指定，它可以通过普通的文本字段和下拉组合区域来使用。

掩码字符串包括两种类型的字符：文字字符，这可以成为输入的一部分；模板字符，这是属于特定类别的占位字符（例如，数字或字母）。例如，下面的代码将一个编辑掩码“（999）999-9999”指定到第一列，该掩码包含的是电话号码，（数字“9”是一个可以代表任何数字的占位字符）：

- Visual Basic

```
' 建立一个电话号码的编辑掩码。
_flex.Cols(1).EditMask = "(999) 999-9999"
```

- C#

```
// 建立一个电话号码的编辑掩码。
_flex.Cols[1].EditMask = "(999) 999-9999";
```

当你将“编辑掩码”属性设置为一个非空的字符串，会导致它必须使用内置的屏蔽编辑，即便该列包含日期/时间值（通常，一个“日期时间提取器”控件可以用于编辑这些列）。如果你有只含时间（没有日期）的“日期时间”列，这使用起来将格外方便。在这些情况下，你可以设置以下属性来使用一个屏蔽编辑器，而不是一个“日期时间提取器”控件：

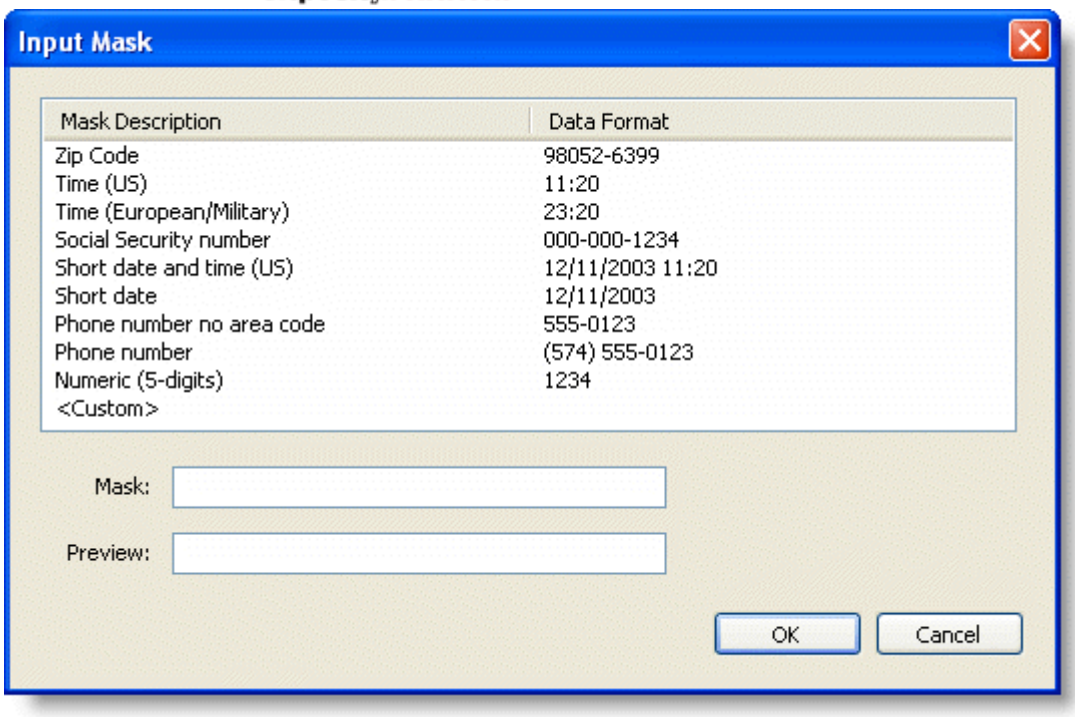
- Visual Basic

```
flex.Cols(1).DataType = GetType(DateTime)
_flex.Cols(1).Format = "hh:mm  tt"
_flex.Cols(1).EditMask = "99:99 LL"
```

- C#

```
flex.Cols[1].DataType = typeof(DateTime);
_flex.Cols[1].Format = "hh:mm  tt";
_flex.Cols[1].EditMask = "99:99 LL";
```

你也可以在设计时通过使用“输入掩码”对话框来设置“编辑掩码”属性。



要访问**输入掩码**对话框可通过以下两种途经，“**列任务**”菜单或 **C1FlexGrid 列编辑器**。

- 进入“列任务”菜单，在编辑掩码框中单击省略号按钮。
- 进入 **C1FlexGrid 列编辑器**，找到左窗格中的**编辑掩码**属性，并单击它旁边的**省略号**按钮。

注意：**输入掩码**对话框是特定的列，并且只会更改所选定的列的**编辑掩码**属性。

有关建立掩码字符串的语法的详细信息，请参阅该控件参考资料部分的编辑掩码属性。

如果在同一列的不同单元格需要不同的掩码，需要捕获编辑前事件，并为当前单元格将编辑掩码属性设置一个的适当的值。

3.6.6 验证

在许多情况下，为确保用户的数据输入是有效的，只有编辑掩码是不够的。例如，掩码不会让你指定一个可能值的范围，或以另一个单元格的内容为基础来验证当前单元格。

在这些情况下，需要捕获验证编辑事件，看看在“编辑器-文本”属性中包含的值是不是当前单元格的有效条目（在这一点上，单元格仍然有它的原始值）。如果该输入的是无效的，将“取消”参数设置为“**真**”，表格将保持编辑模式直到用户输入一个有效的条目。

例如，将下面的代码验证输入到一个货币列，以确保输入的值处于 1000 和

10000 之间：

- Visual Basic

```
Private Sub _flex_ValidateEdit(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.ValidateEditEventArgs) Handles _flex.ValidateEdit
' 验证金额。
If _flex.Cols(e.Col).DataType Is GetType(Decimal) Then
Try
Dim dec As Decimal = Decimal.Parse(_flex.Editor.Text())
If (dec < 1000) Or (dec > 10000) Then
MessageBox.Show("Value must be between 1,000 and 10,000")
e.Cancel = True
End If
Catch
MessageBox.Show("Value not recognized as a Currency")
e.Cancel = True
End Try
End If
End Sub
```

- C#

```
private void _flex_ValidateEdit( object sender, ValidateEditEventArgs e)
{
// 验证金额。
if (_flex.Cols[e.Col].DataType == typeof(Decimal))
{
try
{
Decimal dec = Decimal.Parse(_flex.Editor.Text);
if ( dec < 1000 || dec > 10000 ) {
MessageBox.Show("Value must be between 1,000 and 10,000");
e.Cancel = true;
}
}
catch
{
MessageBox.Show("Value not recognized as a Currency");
e.Cancel = true;
}
}
}
```

3.6.7 自定义编辑器

内置的编辑器可以具备很大的灵活性和能力，但在某些情况下，你可能要使用外部控件作为专门的编辑器。例如，你可能想使用提供了一个下拉计算器的 C1NumericInput 控件来输入数字，或使用一个编辑器来从多列的列表中进行选择，或你可以用自己写的一个专门的控件来编辑你的业务对象。

从基本的“**控件**”类型派生出的任何控件都可以作为一个基本的表格编辑器使用。实现了 **IC1EmbeddedEditor** 接口（在 C1.Common.dll 的定义）的控件可以用表格提供更好的整合和更先进的功能。有关 **IC1EmbeddedEditor** 接口的更多详细信息，请参阅“编辑属性”。

要将一个控件作为自定义编辑器来使用，所有你必须要做的就是使用其“**编辑**”属性将控件的一个实例与一个表格列或一个样式关联到一起。你可以在设计器（使用**列编辑器**）或在代码中实现这步操作。在此之后，表格将自动使用该控件。

要在设计时定义自定义编辑器，先要将编辑器控件的一个实例添加到窗体，然后从 **C1FlexGrid 任务**菜单上选择设计器来打开 **C1FlexGrid 列编辑器**。选择应使用自定义编辑器的列，并将其编辑器的属性设置为新的编辑器控件的名称。

例如，要将 **NumericUpDown** 控件作为一个表格编辑器使用，请按照下列步骤操作：

1. 添加一个 C1FlexGrid 控件到窗体。
2. 将 **NumericUpDown** 控件添加到窗体，并将其“**边框样式**”属性设置为“**无**”。
3. 从 **C1FlexGrid 任务**菜单上选择**设计器**。有关访问 **C1FlexGrid** 列编辑器的更多详细信息，请参阅“访问 C1FlexGrid 的列编辑器”（第 143 页）。
4. 在 C1FlexGrid 列编辑器上，选择第一个滚动的表格列，并将它的“编辑器”属性设置到 NumericUpDown1。

运行该项目并编辑第一列中的一些值。请注意表格如何定位和初始化 NumericUpDown 控件，以便你可以编辑单元格的值。当你完成对一个单元格的编辑，单击不同的单元格或按 TAB 键来移动到下一个。请注意新的值是如何应用到单元格的。

你也可以使用代码来给表格指定自定义编辑器：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 创建自定义编辑器。
Dim editor as New NumericUpDown()
editor.BorderStyle = BorderStyle.None
'  将自定义编辑器分配到表格。
_flex.Cols(1).Editor = editor
End Sub
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
// 创建自定义编辑器。
NumericUpDown editor = new NumericUpDown();
editor.BorderStyle = BorderStyle.None;
// 将自定义编辑器分配到表格。
_flex.Cols[1].Editor = editor;
}
```

创建自定义编辑器

从 Control 基类派生出的任何控件都可以用来作为一个表格编辑器使用。这是完全有可能的，因为在访问诸如“**文本**”和“**界限**”之类的属性，或诸如“**请假**”和“**文本变更**”之类的事件时，表格对基类控件有足够的了解。在很多情况下，这种程度的支持就足够了。

然而，在某些情况下，你可能想使用一些不那么密切地遵守基类的控件。例如，“**日期时间提取器**”控件有一个“**值**”属性，它应该用来检索编辑过的值，而不是文本的值。在这些情况下，你可以在 **IC1EmbeddedEditor** 接口上实现一个或多个方法来覆盖默认的行为。例如，在 C1Input 库中所有控件都支持 **IC1EmbeddedEditor**，因此，它与 C1FlexGrid（还有 **C1TrueDBGrid**）整合密切。

IC1EmbeddedEditor 接口相当简单，并且由于表格是使用后期绑定与之相结合的，你甚至不需要实现其所有成员。只需要落实那个对你的编辑器控件有意义的即可。

接口确实提供了足够的灵活性来使几乎所有的控件都能作为一个表格编辑器使用。你甚至可以使用 UITypeEditor 这些类型来作为表格编辑器。要做到这一点，你需要一个包装类：

1. 派生于 **Control**（UITypeEditor 不可以）。

2. 实现 IC1EmbeddedEditor 接口。
3. 封装相应的 UITypeEditor。

我们在 **CustomEditors** 样品中为这个包装类提供的源代码。

使用 UITypeEditor 包装类，你可以与 C1FlexGrid 一起使用任何 UITypeEditors。.NET 可以为编辑颜色，字体，文件名等提供几个 UITypeEditors。你也可以自己写 UITypeEditors，在某些情况下，这比写控件容易。

可用的示例项目

例如与 C1FlexGrid 一起使用内置的、自定义的和 UITypeEditor 编辑器，请在 ComponentOne 帮助中心上参阅“**自定义编辑器**”样本。

3.6.8 编辑模式

你可以通过阅读编辑属性值来确定表格是否处于编辑模式。如果这个属性恢复为“无效”，则表格不是在编辑模式。不然，属性恢复一个引用值到 正在用于编辑该单元格的控件（该控件可能是一个“文本框”，一个“组合框”，或其他类型的控件），表格就是处于编辑模式。你可以“开始编辑”的方法将表格程式化地设定在编辑模式下，并使用“完成编辑”的方法来完成编辑。

你可以通过处理表格触发的编辑事件来进一步地控制整个编辑过程。在编辑一个单元格的过程中，表格会触发以下一系列事件：

事件	描述
编辑前	一个可编辑的单元格被选中时会触发该事件。你可以通过将事件的“取消”参数设置为“真”来防止单元格被编辑。你还可以修改“组合列表”的属性，以便使适当的下拉按钮被画在单元格中。请注意，用户可能不会在这之后真正开始编辑，他可以简单地将选择移动到一个不同的单元格或控件。
开始编辑	除了用户已经在单元格中敲下了一个键或点击了下拉按钮并真正开始编辑以外，该事件与“编辑前”事件类似。你仍然可以在这一点上取消编辑。请注意，在这一点上“编辑器”属性仍然是空，因为该控件目前还没有确定应该使用的编辑器的类型。您可以在这一点上指定自定义编辑器到“编辑器”属性。
安装编辑器	该事件触发的时间是编辑器控件已经创建并且配置好去编辑该单元格之后，但在它被显示之前。

	在这一点上，你可以改变编辑器的属性（例如，在“文本框”编辑器中设置最大长度或密码字符）。你还可以将自己的事件处理程序添加到编辑器。
验证编辑	该事件触发的时间是当用户完成编辑且编辑器的值被复制回表格之前。你可以通过从表格中检索它来检查原始值（该事件可以提供单元格的坐标）。你可以通过编辑器属性（例如，**Editor.Text**）来检查被分配到表格的新值。如果新的值对单元格无效，请将“*取消*”参数设置为“**真**”，则表格将保持在编辑模式。相反，如果不想将单元格保持在编辑模式下，你最好还是恢复原来的值并离开编辑模式，设置“**取消**”为“**真**”，然后使用“完成编辑”方法。
编辑后	该事件触发的时间是新的值已经被应用到单元格且编辑器已停用之后。你可以使用该事件来更新任何依赖于单元格值的东西（例如，分类汇总或排序）。

3.7 合并单元格

C1FlexGrid 控件允许你合并单元格，使他们跨越多个行或列。这个功能可以用来增强在表格上显示的数据的外观和清晰度。这些设置的作用与 HTML<ROWSPAN>和<COLSPAN>标签类似。

若要使单元格能够合并，以下两件事你必须要做：

1. 将表格 “允许合并”属性设置为“无”以外的任何值。（每个设置的效果都会在参考部分得到解释。）

2. 如果你想要合并列，必须将你想合并的每一列的“允许合并”属性设置为“真”。如果你想要合并行，必须将你想合并的每一行的“允许合并”属性设置为“真”。

如果相邻的单元格中含有相同的非空字符串，合并就会发生。没有一种方法能迫使一对单元格合并到一起。合并会在单元格内容的基础上自动完成。当相邻行中的值呈现重复数据的时候，这可以很容易地提供排序数据的合并后的视图。

单元格合并有几种可能的用途。例如，当文本溢出到相邻列时，你可以用它来创建合并的表头，合并后的数据视图，或表格。

3.7.1 合并表头

要创建合并表头，你必须通过将表格的“允许合并”属性设置到 FixedOnly 来启动。然后，通过设置行和列的“允许合并”属性来指定要合并的行和列。最后，将文本分配到标题单元格，以便你要合并的单元格具有相同的内容。

下面的代码显示了一个例子：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Dim i%
'  初始化该控件。
_flex.Styles.Normal.WordWrap = True
_flex.Cols.Count = 9
_flex.Rows.Fixed = 2
_flex.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
' 创建行标题。
_flex.Rows(0).AllowMerging = True
' 合并含有相同内容的四个单元格。
Dim rng As C1.Win.C1FlexGrid.CellRange = _flex.GetCellRange(0, 1, 0, 4)
rng.Data = "North"
' 合并含有相同内容的四个单元格。
rng = _flex.GetCellRange(0, 5, 0, 8)
rng.Data = "South"
For i = 1 To 4
_flex(1, i) = "Qtr " & i
_flex(1, i + 4) = "Qtr " & i
Next
' 创建列标题。
_flex.Cols(0).AllowMerging = True
' 合并含有相同的内容的两个单元格。
rng = _flex.GetCellRange(0, 0, 1, 0)
rng.Data = "Sales by Product"
' 对齐单元格和自动调整单元格大小。
_flex.Styles.Fixed.TextAlign =
C1.Win.C1FlexGrid.TextAlignEnum.CenterCenter
_flex.AutoSizeCols(1, _flex.Cols.Count - 1, 10)
```

- C#

```
private void Form1_Load(System.object sender, System.EventArgs e)
{
int i;
// 初始化该控件。
_flex.Styles.Normal.WordWrap = true;
_flex.Cols.Count = 9;
_flex.Rows.Fixed = 2;
_flex.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
// 创建行标题。
_flex.Rows[0].AllowMerging = true;
// 合并含有相同内容的四个单元格。
C1.Win.C1FlexGrid.CellRange rng = _flex.GetCellRange(0, 1, 0, 4);
rng.Data = "North";
// 合并含有相同内容的四个单元格。
rng = _flex.GetCellRange(0, 5, 0, 8);
rng.Data = "South";
for ( i = 1 ; i <= 4; i++)
{
_flex[1, i] = "Qtr " + i;
_flex[1, i + 4] = "Qtr " + i;
}
// 创建列标题。
_flex.Cols[0].AllowMerging = true;
// 合并含有相同的内容的两个单元格。
rng = _flex.GetCellRange(0, 0, 1, 0);
rng.Data = "Sales by Product";
// 对齐单元格和自动调整单元格大小。
_flex.Styles.Fixed.TextAlign =
C1.Win.C1FlexGrid.TextAlignEnum.CenterCenter;
_flex.AutoSizeCols(1, _flex.Cols.Count - 1, 10);
}
```

以下是结果：

Sales by Product	North				South			
	Qtr 1	Qtr 2	Qtr 3	Qtr 4	Qtr 1	Qtr 2	Qtr 3	Qtr 4

3.7.2 合并后的数据视图

当表格被绑定到一个数据源时，单元格合并会以同样的方式进行。下面的代码显示了一个在设计时表格绑定到数据源的例子。有关绑定到一个数据源的更多详细信息，请参阅“绑定到一个数据源”（第 95 页）。

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Dim i%
' 设置单元格合并。
_flex.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.RestrictCols
For i = _flex.Cols.Fixed To _flex.Cols.Count - 1
_flex.Cols(i).AllowMerging = True
Next
End Sub
```

- C#

```
private void Form1_Load( System.object sender,        System.EventArgs e)
{
int i;
// 设置单元格合并。
_flex.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.RestrictCols;
for (int i = _flex.Cols.Fixed; i <= _flex.Cols.Count - 1; i++)
{
_flex.Cols(i).AllowMerging = true;
}
```

以下是结果：

Country	City	CompanyName	ContactName
Argentina	Buenos Aires	Cactus Comidas para llevar	Patricio Simpson
		Océano Atlántico Ltda.	Yvonne Moncada
		Rancho grande	Sergio Gutiérrez
Austria	Graz	Ernst Handel	Roland Mendel
	Salzburg	Piccolo und mehr	Georg Pipps
Belgium	Bruxelles	Maison Dewey	Catherine Dewey
	Charleroi	Suprêmes délices	Pascale Cartrain
Brazil	Campinas	Gourmet Lanchonetes	André Fonseca
	Resende	Wellington Importadora	Paula Parente
	Rio de Janeiro	Hanari Carnes	Mario Pontes
		Que Delícia	Bernardo Batista
		Ricardo Adocicados	Janete Limeira
	São Paulo	Comércio Mineiro	Pedro Afonso
		Familia Arquibaldo	Aria Cruz
		Queen Cozinha	Lúcia Carvalho
		Tradição Hipermercados	Anabela Domingues

请注意合并单元格是如何在分组数据和使表格中的信息更容易理解上具有视觉效果的。

可用的示例项目

有关合并后数据视图用 C1FlexGrid 显示的一个示例，请参阅 ComponentOne 帮助中心中的“单元格合并”样本。

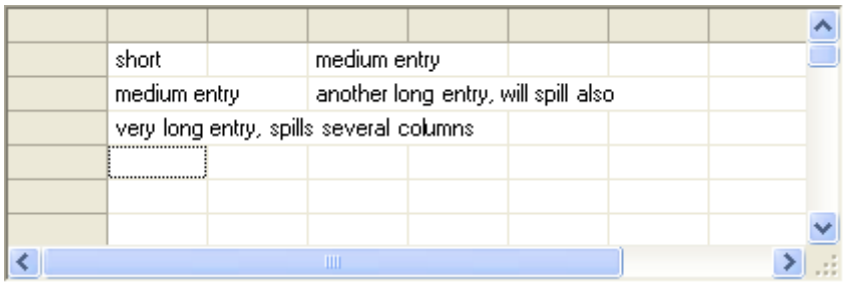
3.7.3 溢出文本

“允许合并”属性有两个设置，它们不同于其他的操作，也不需要你在特定的行和列来设置“允许合并”属性。

溢出设置

溢出设置会导致太长而不适合于一个单元格的文本蔓延到相邻的空单元格中。这种结果式的行为与 Microsoft Excel 中的类似。如果你在一个单元格中键入一个长项而相邻的单元格是空的，则该项可能会溢出所在单元格来占据所需要的尽可能多的空间。

例如，下面的图片显示了当“允许合并”属性设置为“溢出”且用户输入了不同长度的条目时，一个表格看起来是什么样子的：



节点设置

节点设置是与“溢出”类似，但只适用于勾勒节点。当数据被组合成组，且在节点行中所包含的格式中的信息与数据行不同时，此设置是非常有用的。

例如，下面的图片显示了，当数据被用 C1FlexGridBase.Subtotal 方法分组和总结，随后节点被设置为“允许合并”时，一个表格看起来是什么样子的：

此图片与在“创建汇总”（第 67 页）主题中的那个类似。所不同的是，现

在小计行（节点）溢出到相邻的空单元格，改善了表格的外观。

3.7.4 自定义合并

你可以通过两种途径自定义默认的合并行为：

- 将一个自定义的 IComparer 类分配到 CustomComparer 属性。
  - 默认情况下，表格会将包含相同的非空值的相邻的单元格合并到一起。字符串比较是区分大小写的，且包括空白。
  - 如果你想让表格用大小不敏感的比较和修剪空白来合并单元格，你可以写一个自定义的类，用它来实现 Icomparer，并且将它分配到 CustomComparer 属性。
- 写一个从 C1FlexGrid 派生出的新类别，并通过提供自己的自定义合并逻辑来覆盖 GetMergedRange 虚拟方法。

可用的示例项目

有关可以显示如何实现自定义的合并逻辑的样本，请参阅如下样本，“ComponentOne 帮助中心”的“自定义合并”，“自定义合并 2”，“自定义合并 3”，“自定义合并 4”。

3.8 概述和汇总数据

C1FlexGrid 控件具有可以让你汇总数据并以分层方式显示的方法和属性。若要汇总数据并添加聚合值，请使用 C1FlexGrid.Subtotal 方法。若要显示数据的分层视图，请使用“树型”属性。

3.8.1 创建分类汇总

C1FlexGrid.Subtotal 方法可以增加包含普通（非小计）行的汇总数据的分类汇总行。

分类汇总支持分层聚合。例如，如果你的表格包含销售数据，你可能会通过产品、地区和推销员来小计一下以得出总的销售数字。下面的代码说明了这一点：

- Visual Basic

```
Private Sub ShowTotals()
' 在第零列显示大纲栏。
_flex.Tree.Column = 0
_flex.Tree.Style = TreeStyleFlags.Simple
' 清除现有的分类汇总。
_flex.Subtotal(AggregateEnum.Clear)
' 获得总计（使用-1，而不是列索引）。
_flex.Subtotal(AggregateEnum.Sum, -1, -1, 3, "Grand Total")
'  每个产品的总计（第零列）。
_flex.Subtotal(AggregateEnum.Sum,    0,       0,       3,       "Total  {0}")
'  每个区域的总计（第一列）。
_flex.Subtotal(AggregateEnum.Sum,    1,       1,       3,       "Total  {0}")
' 基于内容来调整列宽。
_flex.AutoSizeCols()
End Sub
```

- C#

```
private void ShowTotals()
{
// 在第零列显示大纲栏。
_flex.Tree.Column = 0;
_flex.Tree.Style = TreeStyleFlags.Simple;
// 清除现有的分类汇总。
_flex.Subtotal(AggregateEnum.Clear);
// 获得总计（使用-1，而不是列索引）。
_flex.Subtotal(AggregateEnum.Sum, -1, -1, 3, "Grand Total");
// 每个产品的总计（第零列）。
_flex.Subtotal(AggregateEnum.Sum,    0,       0,       3,       "Total  {0}");
// 每个区域的总计（第一列）。
_flex.Subtotal(AggregateEnum.Sum, 1, 1, 3, "Total {0}");
// 基于内容来调整列宽。
_flex.AutoSizeCols();
}
```

当 C1FlexGrid.Subtotal 方法添加了汇总信息行，它会自动分配汇总样式到新的行（有五个层级的分类汇总内置样式）。你可以使用“**样式编辑器**”或代码在设计器中改变大纲样式的属性，以此来自定义分类汇总行的外观。例如：

- Visual Basic

```
' 设置分类汇总的样式。
Dim cs As C1.Win.C1FlexGrid.CellStyle
cs = _flex.Styles(C1.Win.C1FlexGrid.CellStyleEnum.GrandTotal)
cs.BackColor = Color.Black
cs.ForeColor = Color.White
cs.Font = New Font(Font, FontStyle.Bold)
cs = _flex.Styles(C1.Win.C1FlexGrid.CellStyleEnum.Subtotal0)
cs.BackColor = Color.DarkRed
cs.ForeColor = Color.White
cs.Font = New Font(Font, FontStyle.Bold)
cs = _flex.Styles(C1.Win.C1FlexGrid.CellStyleEnum.Subtotal1)
cs.BackColor = Color.DarkBlue
cs.ForeColor = Color.White
```

- C#

```
// 设置分类汇总的样式。
CellStyle cs;
cs = _flex.Styles[CellStyleEnum.GrandTotal];
cs.BackColor = Color.Black;
cs.ForeColor = Color.White;
cs.Font = new Font(Font, FontStyle.Bold);
cs = _flex.Styles[CellStyleEnum.Subtotal0];
cs.BackColor = Color.DarkRed;
cs.ForeColor = Color.White;
cs.Font = new Font(Font, FontStyle.Bold);
cs = _flex.Styles[CellStyleEnum.Subtotal1];
cs.BackColor = Color.DarkBlue;
cs.ForeColor = Color.White;
```

执行此代码后，表格看起来是这样的：

总计行中包含的所有产品，地区和销售人员的销售总额。它是使用 *groupOn* 参数-1 在调用 C1FlexGrid.Subtotal 方法时被创建的。其他分类汇总显示产品和地区的销售总额。他们是用 Groupon 的参数值 0 和 1 创造的。

除了总量之外，你也可以计算其他分类汇总（例如，平均值或百分比），并计算每一行的几个汇总（例如，毛销售额及净销售额）。

由**分类汇总**的方法创建的小计行不同于其他普通行，主要体现在三个方面：

1. 以 *flexSTClear* 参数来调用**分类汇总**的方法，小计行可以被自动删除。在用户可以移动列并对数据进行重新排序的地方，重新计算分类汇总很有必要，而这在提供数据的动态视图方面非常有用。
2. 小计行可作为一个大纲中的节点使用，它可以让你折叠和展开行组来展现数据的概述或透露其细节。要看到大纲的树型图，你需要设置列和“树型样式”的属性来确定大纲树型图的位置和外观。
3. 小计行可以被视为树型结构上的的节点。通过“节点”属性，你可以为任何小计行获得一个“节点”对象。
4. 当表格被绑定到一个数据源，小计行并不符合实际的数据。如果你在数据源中移动光标，则小计行将在表格中被跳过。

大纲树型图可以允许用户通过点击节点来折叠和展开表格的各部分。你可以使用大纲树型图来显示许多类型的信息，不仅仅是汇总这一种。下一主题将会指出如何创建一个自定义的大纲树型图来显示目录信息。

3.8.2 创建自定义树型图

要想不使用分类汇总法来创建大纲树型图，你需要遵循以下步骤：

1. 将行添加到表格中。
2. 通过将他们的“是节点”属性设置为“真”，将一些行转变成大纲的节点。
3. 获得每个节点行的节点对象，并设置其“等级”属性来确定树型层次结构中节点的位置值越高意味着该节点在大纲树型图中越深入（缩进得多）。

例如，下面的代码可以创建一个目录树：

- Visual Basic

```
' 在窗体的顶部添加这些输入声明。
Imports System.IO
Imports C1.Win.C1FlexGrid
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 初始化表格布局。
_flex.Cols.Fixed = 0
_flex.Cols.Count = 1
_flex.Rows.Count = 1
_flex.ExtendLastCol = True
_flex.Styles.Normal.TextAlign = TextAlignEnum.LeftCenter
_flex.Styles.Normal.Border.Style = BorderStyleEnum.None
' 显示大纲树型图。
_flex.Tree.Column = 0
_flex.Tree.Style = TreeStyleFlags.SimpleLeaf
_flex.Tree.LineColor = Color.DarkBlue
' 填充表格。
AddDirectory("c:\", 0)
End Sub
```

- C#

```
// 在窗体的顶部添加这些输入声明。
using       System;
using       System.Collections.Generic;
using       System.ComponentModel;
using       System.Data;
using       System.Drawing;
using       System.Text;
using       System.Windows.Forms;
using       System.IO;
using       C1.Win.C1FlexGrid;
private void Form1_Load(object sender, EventArgs e)
{
// 初始化表格布局。
_flex.Cols.Fixed = 0;
_flex.Cols.Count = 1;
_flex.Rows.Count = 1;
_flex.ExtendLastCol = true;
_flex.Styles.Normal.TextAlign = TextAlignEnum.LeftCenter;
_flex.Styles.Normal.Border.Style = BorderStyleEnum.None;
// 显示大纲树型图。
_flex.Tree.Column = 0;
_flex.Tree.Style = TreeStyleFlags.SimpleLeaf;
_flex.Tree.LineColor = Color.DarkBlue;
// 填充表格。
AddDirectory(@"c:\\", 0);
}
```

以上的代码初始化了表格布局，并且调用了“添加目录”程序，这意味着做了填充网格和建立的树型结构的工作：

- Visual Basic

```
Private Sub AddDirectory(ByVal dir As String, ByVal level As Integer)
' 添加该目录。
Dim thisDir As String
thisDir = Path.GetFileName(dir)
If thisDir.Length = 0 Then thisDir = dir
_flex.AddItem(thisDir)
' 使这个新行成为一个节点。
Dim row As Row
row = _flex.Rows(_flex.Rows.Count - 1)
row.IsNode = True
' 设置该节点的层级。
Dim nd As Node
nd = row.Node
nd.Level = level
' 在此目录中添加文件。
Dim file As String, cnt As Integer, r As Row
cnt = 0
For Each file In Directory.GetFiles(dir)
_flex.AddItem(Path.GetFileName(file).ToLower())
r = _flex.Rows(_flex.Rows.Count – 1)
r.IsNode = True
r.Node.Level = level + 1
cnt = cnt + 1
If cnt > 10 Then Exit For
Next
' 添加子目录（到 4 级）。
If level <= 4 Then
Dim subdir As String
cnt = 0
For Each subdir In Directory.GetDirectories(dir)
AddDirectory(subdir, level + 1)
cnt = cnt + 1
If cnt > 10 Then Exit For
Next
```

- C#

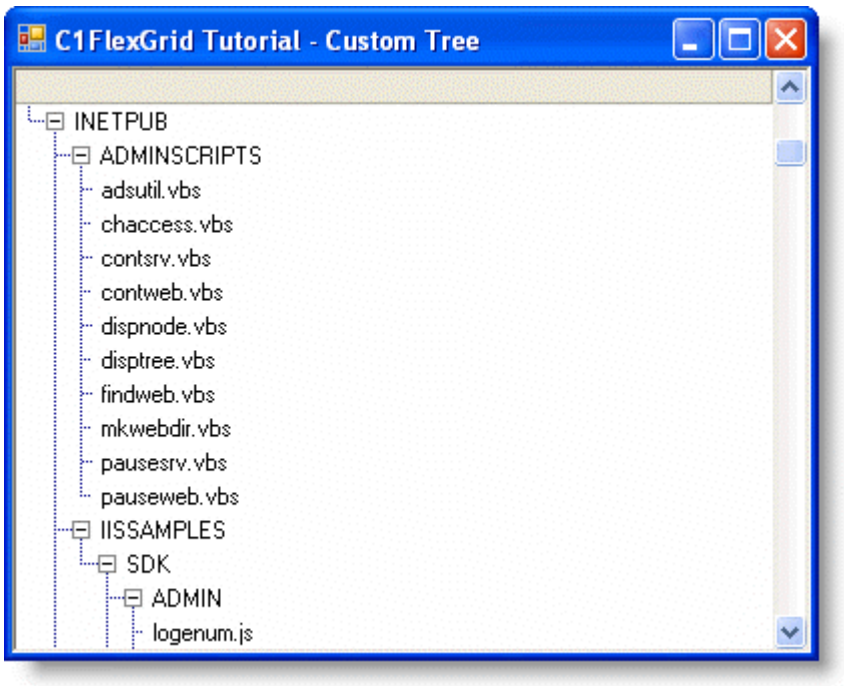
“添加目录”是一个递归程序，它横跨当前目录及其所有子目录。在这个

```
private void AddDirectory(string dir, int level)
{
// 添加该目录。
string thisDir = Path.GetFileName(dir);
if (thisDir.Length == 0) { thisDir = dir; }
_flex.AddItem(thisDir);
// 使这个新行成为一个节点。
Row row = _flex.Rows[_flex.Rows.Count - 1];
row.IsNode = true;
// 设置该节点的层级。
Node       nd = row.Node;
nd.Level = level;
// 在此目录中添加文件。
int cnt = 0;
Row r;
foreach (string file in Directory.GetFiles(dir))
{
_flex.AddItem(Path.GetFileName(file).ToLower());
// 将没有子行的行标记为节点。
r = _flex.Rows[_flex.Rows.Count - 1];
r.IsNode = true;
r.Node.Level = level + 1;
cnt = cnt + 1;
if (cnt > 10) break;
}
// 添加子目录（到 4 级）。
if (level <= 4)
{
cnt = 0;
foreach (string subdir in Directory.GetDirectories(dir))
{
AddDirectory(subdir, level + 1);
cnt = cnt + 1;
if (cnt > 10) break;
}
}
}
```

例子中，为了节省时间，树的大小局限在四个目录层次。在实际应用中，当它们被扩大时该程序应改为仅填充树型分支（请参阅式 FlexGrid for WinForms 教

程（第 107 页））。

此代码可以创建一个表格，看起来如下图：



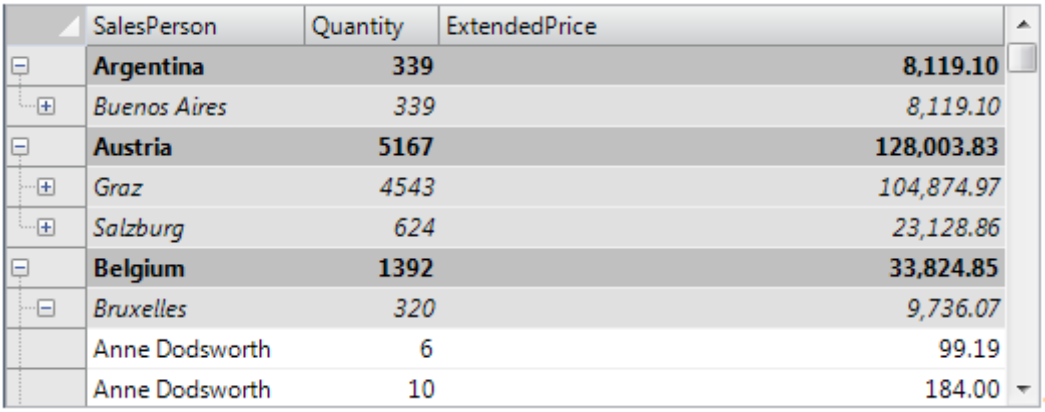
3.8.3 用 C1FlexGrid 控件来创建大纲和树型图

C1FlexGrid 控件独特的和流行的特点之一是能够添加层次分组到常规的非结构化数据。

	Country	City	SalesPerson	Quantity	ExtendedPrice
	Argentina	Buenos Aires	Andrew Fuller	7	322.00
	Argentina	Buenos Aires	Andrew Fuller	20	155.00
	Argentina	Buenos Aires	Anne Dodsworth	1	12.50
	Argentina	Buenos Aires	Anne Dodsworth	2	527.00
	Argentina	Buenos Aires	Anne Dodsworth	5	405.00
	Argentina	Buenos Aires	Janet Leverling	12	96.00
	Argentina	Buenos Aires	Janet Leverling	12	223.20
	Argentina	Buenos Aires	Laura Callahan	3	54.00
	Argentina	Buenos Aires	Laura Callahan	6	75.00

为了实现这一目标，C1FlexGrid 介绍了节点行的概念。节点行不包含常规的数据。相反，他们作为表头在类似数据的分组下面运作，酷似一个常规 **TreeView** 控件中的节点。就像一个 **TreeView** 控件中的节点，节点行可以折叠和扩展，隐藏或显示它们所包含的数据。像一个 **TreeView** 控件中的节点的另一方面是，节点行有一个能定义节点层次的层级属性。较低级别的节点包含较高级别的节点。例如，假设你有一个可以显示客户名称，国家，城市，销售额的表格。这种典型的表格通常看起来是这样的

k



	SalesPerson	Quantity	ExtendedPrice
⊟	**Argentina**	**339**	**8,119.10**
⊞	*Buenos Aires*	*339*	*8,119.10*
⊟	**Austria**	**5167**	**128,003.83**
⊞	*Graz*	*4543*	*104,874.97*
⊞	*Salzburg*	*624*	*23,128.86*
⊟	**Belgium**	**1392**	**33,824.85**
⊟	*Bruxelles*	*320*	*9,736.07*
	Anne Dodsworth	6	99.19
	Anne Dodsworth	10	184.00

所有的信息都是存在的，但很难看到每一个国家或客户的总销售额。你可以使用 C1FlexGrid 的概述功能按国家（0 级）对数据进行分组，然后按每个国家的城市（1 级），然后按每个城市的顾客（2 级）。下面是加入大纲之后的相同表格：

该表格会像前一个（被绑定到同一数据源的）一样显示相同的信息，但它增加了一个树型图，那里的每个节点都包含了它下面的数据摘要。节点可以折叠起来只显示摘要，或展开以显示细节。请注意，每个节点一行都可以显示多个列的摘要（在这种情况下，合计单位销售量并合计总金额）。

在这篇文章中，我们将引导你熟悉将一个普通的表格转变成一个更丰富的大纲型表格的过程。

加载数据

将数据加载到一个大纲型表格与将其加载到一个普通的表格是完全相同的。如果你的数据源在设计时是可用的，你可以使用 Visual Studio 属性窗口来设置表格的“数据源”属性，并且无需编写任何代码就可以将表格绑定到数据。

如果数据源在设计时是不可用的，你可以在代码中设置表格的“数据源”属性。数据绑定代码看起来通常是这样的：

代码使用一个 **OleDbDataAdapter** 来用数据填充一个**数据表**，然后，将**数据表**分配给表格的“**数据源**”属性。

运行此代码后，你会看到在第一张图片中显示了一个“普通的表格”。要使这个普通的表格变成第二张图片中显示的那种大纲型表格，我们需要插入节点行来整理这个大纲。

创建节点行

节点行几乎都是同样的普通行，但以下情况除外：

```
public Form1()
{
InitializeComponent();
// 获取数据
var fields = @"
Country,
City,
SalesPerson,
Quantity,
ExtendedPrice";
var sql = string.Format("SELECT {0} FROM Invoices ORDER BY {0}",
fields);
var da = new OleDbDataAdapter(sql, GetConnectionString());
da.Fill(_dt);
// 将表格绑定到数据
this._flex.DataSource = _dt;
// 为“总价”列设置格式
_flex.Cols["ExtendedPrice"].Format = "n2";
}
```

- 节点行不是数据绑定。当表格被绑定到一个数据源时，每个普通的行会对应数据源中的一个项目。而节点行则不然。相反，它们的存在是为了给包含类似数据的普通行分组。
- 节点行可以折叠或展开。当一个节点行折叠起来时，它的所有数据和子节点都隐藏起来了。如果大纲树型图可见，用户可以用鼠标或键盘来折叠和展开节点。

如果大纲树型图不可见，则只能用代码来扩展或折叠节点。

你可以使用“是节点”属性来确定一个行是否是节点行：

节点行可以用以下三种方法来创建：

1. 使用 **Rows.InsertNode** 方法。这将在指定的索引中插入一个新的节点行。一旦该节点行被创建成功，你可以像使用任何其他行一样使用它（设置每列的数据、应用样式等）。
2. 使用 C1FlexGrid.Subtotal 方法。这种方法会在表格中数据发生变化的地方用可选的分类汇总来扫描整个表格并自动插入节点行。这是插入汇总和构建大纲的“高层次”方式。它只需要非常少的代码，但对表格中的数据是如何排列的和大纲看起来应该像什么样子做了一些相关的假设。

```
var row = _flex.Rows[rowIndex];
if (row.IsNode)
{
// 该行是一个节点
var node = row.Node;
DoSomethingWithTheNode(node);
}
else
{
// 该行不是一个节点

}
```

3. 如果表格是未绑定的，那么你可以通过将 “是节点”属性设置为“真”来将一些普通行变成节点行。请注意，这仅限于当表格处于未绑定的状态下。试图将一个普通的数据绑定行变成一个节点，可能会造成表格抛出一个异常。

以下的代码演示了你应该如何来执行一个“分组依据”程序，插入节点行并对一个给定列的近似值进行分组。

```
// 在给定列上将插入的同一个给定级别的节点分组
void GroupBy(string columnName, int level)
{
object current = null;
for (int r = _flex.Rows.Fixed; r < _flex.Rows.Count; r++)
{
if (!_flex.Rows[r].IsNode)
{
var value = _flex[r, columnName];
if (!object.Equals(value, current))
{
// 值的变化：插入节点
_flex.Rows.InsertNode(r, level);
// 在第一个滚动列显示分组的名称
_flex[r, _flex.Cols.Fixed] = value;
// 更新当前值
current = value;
}
}
}
}
```

代码可以跳过现有的节点行（因此它可以被称为添加几个层次的节点）来扫描所有列，并记录分组列的当前值的轨迹。当当前值发生变化时，在第一滚动列会插入一个节点行，新组的名称会在此显示。

回到我们的例子，你可以使用此方法通过调用来创建两个级别的大纲：

```
void _btnGroupCountryCity_Click(object sender, EventArgs e)
{
GroupBy("Country", 0);
GroupBy("City", 1);
}
```

这很简单，但也有一些需要注意的事项。首先，该方法假定数据是按照大纲结构进行排序的。在这个例子中，如果数据是按照**销售人员**，而不是按照**国别**排序的，那么该大纲中每个国家都会对应好几个零级节点，这可能并不是你想要的。此外，“**分组依据**”程序可以插入许多行，这可能导致表格闪烁不定。为了避免这种情况，通常你应该先将“重绘”属性设置为“假”，然后再作出更新，并且当完成后再将其设置回“真”。

为了解决这些问题，用来创建大纲的代码应该重新编写如下：

```
void _btnGroupCountryCity_Click(object sender, EventArgs e)
{
// 暂停重绘，同时更新
using (new DeferRefresh(_flex))
{
// 恢复原来的排序（按照国家、城市等）
ResetBinding();
// 按照国家、城市来分组
GroupBy("Country", 0);
GroupBy("City", 1);
}
}
```

“**延迟刷新**”类是一种简单的功能，它可以将表格的“重绘”属性设置为“假”，并且当它被破坏时可以恢复其原有的值。这将确保“重绘”属性 被完全地恢复，即便是在更新时发生例外的情况下。以下是 “**延迟刷新**”类的执行情况：

```
/// 实用工具类，用于封装在重绘版块的表格冗长的操作。
/// 这样就避免了闪烁，并且可以确保在发生万一的情况下“重绘”属性能够妥善地复位。
/// 在操作过程中抛出一个异常。
class DeferRefresh : IDisposable
{
C1FlexGrid _grid;
bool _redraw;
public DeferRefresh(C1FlexGrid grid)
{
_grid = grid;
_redraw = grid.Redraw;
grid.Redraw = false;
}
public void Dispose()
{
_grid.Redraw = _redraw;
}
}
```

“绑定表格”的方法可以确保表格会按照我们的大纲结构所需的顺序排序。在我们的例子中，排序的顺序是按照国家、城市和销售人员。代码看起来则是像这个样子的：

如果你现在运行此代码，你会发现该节点行如预期一样创建，但大纲树型图是不可见的，所以你不能展开和折叠节点。下一节会对大纲树型图进行描述。

```
// unbind and re-bind grid in order to reset everything
void ResetBinding()
{
// 解除绑定表格
_flex.DataSource = null;
// 重置任何自定义排序
_dt.DefaultView.Sort = string.Empty;
// 重新绑定表格
_flex.DataSource = _dt;
// 设置总价列的格式
_flex.Cols["ExtendedPrice"].Format = "n2";
// 自动调整列宽以适配其内容
flex.AutoSizeCols();
}
```

3.8.4 大纲树型图

大纲树型图与你在一个普通的“**树型视图**”控件中看到的非常相似。它显示了一个在每个节点行旁边有折叠或扩展的图标的缩进结构，以便用户可以展开和折叠大纲以看到理想层级的细节。

大纲树型图可以在任何列中显示，这取决于由 **Tree.Column** 属性。在默认情况下，这个属性被设置为-1，这能导致大纲树型图根本不会被显示。在上面给出的例子中，为了显示大纲树型图，你可以使用这样的代码：

```
void _btnTreeCountryCity_Click(object sender, EventArgs e)
{
using (new DeferRefresh(_flex))
{
// 像以前一样按照国家和城市分组
_btnGroupCountryCity_Click(this, EventArgs.Empty);
// 显示大纲树型图
_flex.Tree.Column = 0;
// 自动调整尺寸以容纳树
_flex.AutoSizeCol(_flex.Tree.Column);
// 折叠细节节点
_flex.Tree.Show(1);
}
}
```

代码调用先前的方法来建立大纲，然后再将 **Tree.Column** 属性设置为零，以便在第一列显示大纲树型图。它还调用 C1FlexGrid.AutoSizeCol 的方法，以确保该列足够宽，能够容纳大纲树型图。最后，它调用 **Tree.Show** 的方法来显示所有零级节点（在这种情况下，就是城市那一层）并隐藏所有的细节。

- “树”的属性使一个引用返回到公开了可以用于定制大纲树型图的几种方法和属性的 GridTree 对象，。其中主要的一些列举如下：
- **列**：获取或设置包含了大纲树型图的列的索引。将此属性设置为-1 可以导致用户看不到大纲树型图。
- **缩进**：以像素为单位，在相邻的节点层级之间，获取或设置缩进。更高的缩进层级会导致树变得越来越宽。
- **样式**：获取或设置即将显示的大纲树型图的样式。用这个属性来判断该树型图是否应包括顶部的一个按钮栏，来允许用户折叠/展开整个树型

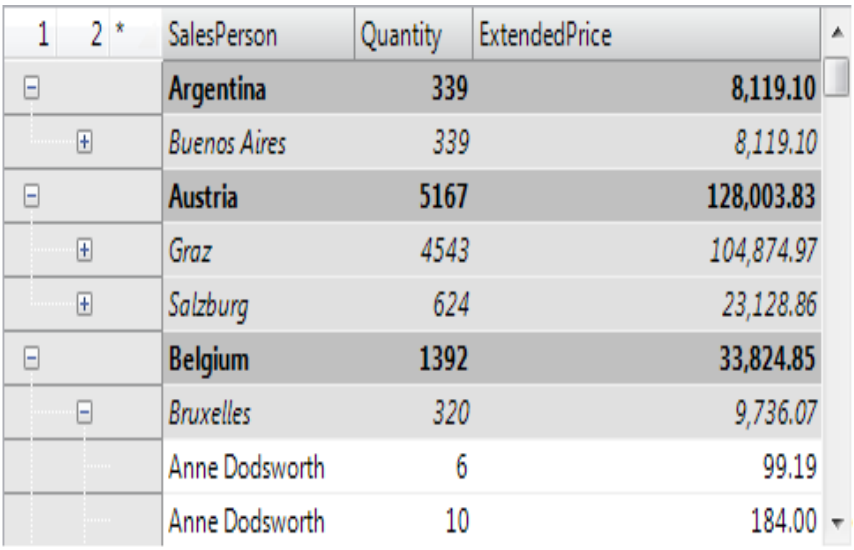
图，来判断线条和（或）符号是否应被显示，来判断线条是否应被显示既连接到了数据行又连接到了节点行。

- **线条颜色**：获取或设置树型图连接线的颜色。
- **线条样式**：获取或设置树型图连接线的样式。

例如，通过改变上面的代码来包含这两个线条：

```
// 显示大纲树型图
_flex.Tree.Column = 0;
_flex.Tree.Style = TreeStyleFlags.CompleteLeaf;
_flex.Tree.LineColor = Color.White;
_flex.Tree.Indent = 30;
```

大纲树型图会改变如下：



1 2 *	SalesPerson	Quantity	ExtendedPrice
⊟	**Argentina**	**339**	**8,119.10**
⊞	*Buenos Aires*	*339*	*8,119.10*
⊟	**Austria**	**5167**	**128,003.83**
⊞	*Graz*	*4543*	*104,874.97*
⊞	*Salzburg*	*624*	*23,128.86*
⊟	**Belgium**	**1392**	**33,824.85**
⊟	*Bruxelles*	*320*	*9,736.07*
	Anne Dodsworth	6	99.19
	Anne Dodsworth	10	184.00

请注意标有“1”，“2”和“*”的、在左上角单元格上的按钮。点击这些按钮，会导致整个树型图折叠或展开到相应的级别。还请注意，更宽的缩进和线条将树型图既连接到普通行（“Anne Dodsworth”）又连接到节点行。

3.8.5 添加分类汇总

到目前为止，我们已经介绍了节点行和大纲树型图的创建。然而，为了使大纲真正有用，节点行应该包括它们所包含的数据的汇总信息。

如果你使用 C1FlexGrid.Subtotal 方法来创建一个大纲树型图，然后分类汇总将会被自动添加。这将在后面的章节中描述。

如果你像上面所描述的使用 **Rows.InsertNode** 方法来创建大纲树型图，那么你应该使用 C1FlexGrid。总结用来计算每个行的分组中的分类汇总方法，并且将结果直接插入节点行。

下面列举出来的 C1FlexGrid.Subtotal 方法演示了如何做到这一点：

```
// 在一个给定层级的每个节点添加分类汇总
void AddSubtotals(int level, string colName)
{
// 获取我们将要进行汇总的列
int colIndex = _flex.Cols.IndexOf(colName);
// 为找到合适层级的节点对行进行扫描
for (int r = _flex.Rows.Fixed; r < _flex.Rows.Count; r++)
{
if (_flex.Rows[r].IsNode)
{
var node = _flex.Rows[r].Node;
if (node.Level == level)
{
// 找到一个节点，计算总价的总和
var range = node.GetCellRange();
var sum = _flex.Aggregate(AggregateEnum.Sum,
range.r1, colIndex, range.r2, colIndex,
AggregateFlags.ExcludeNodes);
// 在表格上显示总和
// （将自动使用列格式）
_flex[r, colIndex] = sum;
}
}
}
}
```

“**添加分类汇总**”方法可以扫描所有表格中的行来寻找节点行。当一个理想层级的节点行被发现时，该方法可以使用“获取单元格区域”的方法来检索节点的子行。然后它使用 C1FlexGrid.Aggregate 方法来计算整个范围内目标列中

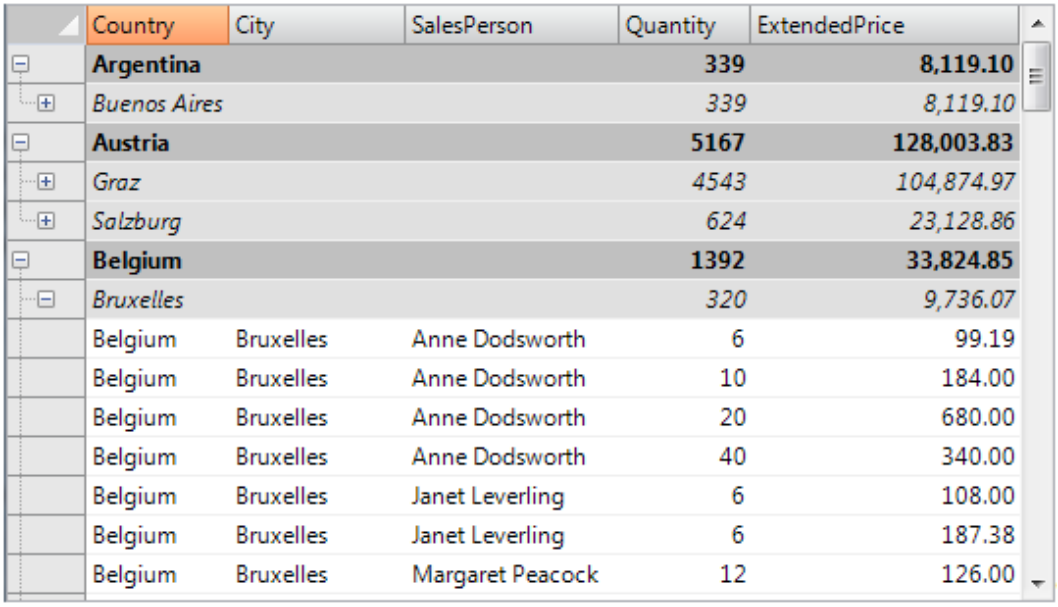
值的总和。**聚合**的调用包括一个可以避免重复计算的现有节点的“**排除节点**”的标志。一旦小计已经被计算，它就会与通常的_**flex[行，彩色]**索引一起被分配到节点行的单元格。

请注意，这并不会以任何方式影响到数据源，因为节点行并没有绑定到数据。还要注意，该方法可以被用来将多个汇总添加到每个节点行。在这个例子中，我们将增加数量和总价列的汇总。除了总金额，你还可以添加其他的集合值，比如平均值，最高值，最低值等等。

现在我们可以用这种方法来创建一个包括节点行、大纲树型图以及分类汇总的完整的大纲：

```
void _btnTreeCountryCity_Click(object sender, EventArgs e)
{
using (new DeferRefresh(_flex))
{
// 恢复原来的排序（按照国家、城市、销售人员）
ResetBinding();
// 按照国家、城市分组
GroupBy("Country", 0); // 按 country (level 0) 分组
GroupBy("City", 1);        // 按 city (level 1) 分组
// 添加每个国家、城市的总数
AddSubtotals(0,  "ExtendedPrice");      //extended price per
country   (level   0)
AddSubtotals(0,  "Quantity");   // quantity per country
(level      0)
AddSubtotals(1,  "ExtendedPrice");      // extended price per city
(level      1)
AddSubtotals(1,  "Quantity");   // quantity per city (level 1)
// 显示大纲树型图
_flex.Tree.Column = 0;
_flex.AutoSizeCol(_flex.Tree.Column);
_flex.Tree.Show(1);
}
}
```

如果你现在运行该项目，你会看到一个由节点行组成的树型图，其中显示了每个国家和城市的销售总数量和金额。虽然这非常好，但还是有一个小问题。如果你展开任何节点行，你会看到大量的重复值。一个给定的城市节点下的所有行都有相同的国家和城市：

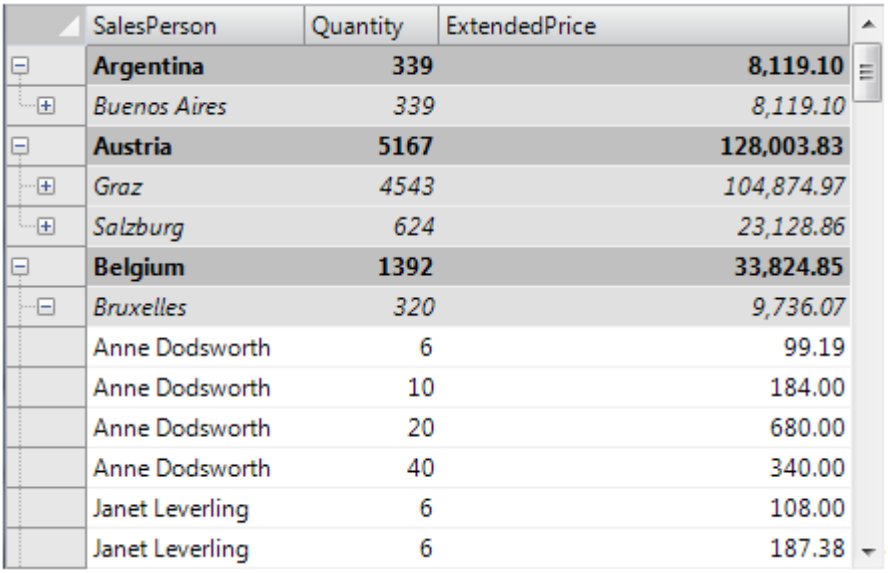


Country	City	SalesPerson	Quantity	ExtendedPrice
Argentina			**339**	**8,119.10**
Buenos Aires			*339*	*8,119.10*
Austria			**5167**	**128,003.83**
Graz			*4543*	*104,874.97*
Salzburg			*624*	*23,128.86*
Belgium			**1392**	**33,824.85**
Bruxelles			*320*	*9,736.07*
Belgium	Bruxelles	Anne Dodsworth	6	99.19
Belgium	Bruxelles	Anne Dodsworth	10	184.00
Belgium	Bruxelles	Anne Dodsworth	20	680.00
Belgium	Bruxelles	Anne Dodsworth	40	340.00
Belgium	Bruxelles	Janet Leverling	6	108.00
Belgium	Bruxelles	Janet Leverling	6	187.38
Belgium	Bruxelles	Margaret Peacock	12	126.00

这是正确的，但它同时也是一种 屏幕实际使用面积的浪费。消除这些重复的值是很容易的；所有你所需要做的只是将正在被分组的列的**宽度**设置为零。然而，当你这样做时，你应该记得将表格的“允许合并”属性设置为“节点”，以便使分配给节点行的文本可以溢出到可见列。（另一种选择是将节点的文本指定到第一个可见列，但合并通常是一个更好的解决方案，因为它可以让你在节点行使用更长的文本）。

下面是修改后的代码和最终结果：

```
        void _btnTreeCountryCity_Click(object sender, EventArgs e)
        {
        using (new DeferRefresh(_flex))
        {
        // 恢复原来的排序（按照国家、城市、销售人员）
        ResetBinding();
        // 按照国家、城市来分组
        GroupBy("Country", 0); // group by country (level 0)
        GroupBy("City", 1);        // group by city (level 1)
        // 隐藏我们分过组的那些列
        //（他们只有在树型图节点上已经出现过的那些重复的值）
        //（但不要使它们不可见，这可能会隐藏节点的文本）
_flex.Cols["Country"].Width = 0;
        _flex.Cols["City"].Width = 0;
        // 允许节点的内容溢出到下一个单元格
        _flex.AllowMerging = AllowMergingEnum.Nodes;
        // 添加每个国家和城市的总数
        AddTotals(0,      "ExtendedPrice");      //      extended      price    per
           country
        (level      0)
        AddTotals(0,      "Quantity");    //      quantity        per      country
           (level
        0)
        AddTotals(1,      "ExtendedPrice");      //      extended      price    per
           city
        (level      1)
        AddTotals(1,      "Quantity");    //      quantity        per      city     (level
           1)
        // 显示大纲树型图
        _flex.Tree.Column = 0;
        _flex.AutoSizeCol(_flex.Tree.Column);
        _flex.Tree.Show(1);
        }
        }
```



	SalesPerson	Quantity	ExtendedPrice
⊟	**Argentina**	**339**	**8,119.10**
⊞	*Buenos Aires*	*339*	*8,119.10*
⊟	**Austria**	**5167**	**128,003.83**
⊞	*Graz*	*4543*	*104,874.97*
⊞	*Salzburg*	*624*	*23,128.86*
⊟	**Belgium**	**1392**	**33,824.85**
⊟	*Bruxelles*	*320*	*9,736.07*
	Anne Dodsworth	6	99.19
	Anne Dodsworth	10	184.00
	Anne Dodsworth	20	680.00
	Anne Dodsworth	40	340.00
	Janet Leverling	6	108.00
	Janet Leverling	6	187.38

国家和**城市**列现在是不可见的，但它们的值仍然出现在节点行。折叠起来的的树型图可以显示每个国家和城市的总数。

3.8.6 使用分类汇总方法

在前面我们曾经提到，你也可以使用 **C1FlexGrid** 的 C1FlexGrid.分类汇总方法来创建树型图。除了可以在一个单一的步骤里同时做两件事以外，这种分类汇总的方法也可以像以上所描述的“**分组依据**”和“**添加分类汇总**”一样来执行相同的任务，因此它比其他的更高效一些。

下面的代码显示了你应该如何使用**分类汇总**的方法来完成我们之前做过的同样的事情，只是要比之前更快一点而且不需要使用任何辅助方法：

```
void _btnTreeCountryCity_Click(object sender, EventArgs e)
{
using (new DeferRefresh(_flex))
{
// 恢复原来的排序（按照国家、城市、销售人员）
ResetBinding();
// group and total by country and city
_flex.Subtotal(AggregateEnum.Sum,    0,      "Country",
"ExtendedPrice");
_flex.Subtotal(AggregateEnum.Sum,    0,      "Country",     "Quantity");
_flex.Subtotal(AggregateEnum.Sum,    1,      "City", "ExtendedPrice");
_flex.Subtotal(AggregateEnum.Sum,    1,      "City", "Quantity");
// 隐藏我们分过组的那些列
//（他们只有在树型图节点上已经出现过的那些重复的值）
//（但不要使它们不可见，这可能会隐藏节点的文本）
_flex.Cols["Country"].Width = 0;
_flex.Cols["City"].Width = 0;
_flex.AllowMerging = AllowMergingEnum.Nodes;
// 显示大纲树型图
_flex.Tree.Column = 0;
_flex.AutoSizeCol(_flex.Tree.Column);
_flex.Tree.Show(1);
}
}
```

分类汇总的方法非常方便和灵活。它包含多个重载，能够使你明确应该对哪些列进行分组并按照索引或名称来合计总数，是否包括它插入节点行的标题，以及如何进行分组等等。下面的摘要介绍了可用的重载：

1. Subtotal（AggregateEnum aggType）

该方法的这个版本只有一个聚合类型作为一个参数。只有消除现有的分类汇总，然后再插入新的，它才是有用的。在这种情况下， aggType 参数要设置

到 AggregateEnum.Clear。

2. Subtotal(AggregateEnum aggType, int groupBy, int totalOn)
Subtotal(AggregateEnum aggType, string groupBy, string totalOn)

这都是最常用的重载。参数都是聚合类型的，可以用来插入和分组和计算总数的列。列是可以通过索引或名称来引用的。而后者则是我们在上面的例子中使用过的。

3. **Subtotal(**AggregateEnum aggType, int groupBy, int totalOn**, string caption)**
Subtotal(AggregateEnum aggType, string groupBy, string totalOn**, string caption)**

这些重载添加了一个额外的标题参数。标题参数明确了被添加到新的节点行的文本，以确定要进行分组的值。默认情况下，要分组的值会显示出来，所以如果你是按国家分组的，节点行会显示“阿根廷”，“巴西”，等等。如果你设置的标题参数为字符串，如“国家：{0}”，那么该节点的行会相应地显示“国家：阿根廷”来代替。

4. **Subtotal**(AggregateEnum aggType, **int groupFrom**, **int groupTo**, int totalOn, string caption)
Subtotal(AggregateEnum aggType, **string groupFrom**, **string groupTo**, string totalOn, string caption)

这些重载将 groupby 函数参数分成了两个：groupFrom 和 groupTo。在默认情况下，无论什么时候 GroupBy 的值或以往的任何列发生了变化，分类汇总的方法都可以插入一个节点行。例如，即使 GroupBy 列的值是相同的，如果某一行在“城市”这一列有与前行中相同的值，但在“国家”一栏有不同的值，那么分类汇总的方法也会认为行应该在不同的分组里面并且插入一个新的节点行。这些聚合使你撤销该行为，并且指定在确定一组时应该考虑的列的范围。

3.8.7 大纲维护

到目前为止，我们已经讨论了如何使用高级别的 C1FlexGrid. Subtotal 方法以及较低级别的 **Rows.InsertNode** 和 **Aggregate** 方法来创建总计包含树型图和总计的大纲。

在这一点上，重要的是要记住，大纲树型图是基于数据而创建的，但是它不被任何方式约束，并且当表格或数据有变化时它不能自动维持现状。例如，如果用户在“总价”这一列修改了一个值，则分类汇总不会进行自动更新。如果用户将表格进行排序，那么数据将会被刷新，且分类汇总会消失。

有两种常见的方式可以用来维持大纲：

- 防止用户做任何将会使大纲无效的更改。这是最简单的选择。你可以设置表格的“允许编辑”、“允许拖动”和“允许排序”属性为“假”，并且阻止任何会影响到大纲的更改。
- 当数据或表格有变化时更新大纲时。你会附加表格的“数据刷新后”、“排序后”

和“编辑后”事件的处理程序，并重新生成合适的大纲。

选项二通常是更有趣的，因为它提供了一个快捷而且简单的动态数据分析工具。这种方法是 C1FlexGrid 提供的“**分析**”示例来说明的。该示例创建了一个初始大纲，并允许用户对列进行重新排序。当列的顺序发生变化时，该示例自动对数据进行重新排序，并重新创建大纲。用户可以方便地创建简单的报告，来按国家、按产品、按销售人员等显示销售。

3.8.8 使用节点类

“节点”类提供了许多可以用来创建和管理大纲树型图的方法和属性。这些方法和属性中的许多都是基于标准的 TreeView 对象模型，因此他们应该熟悉大多数的开发者。

为了获得一个“节点”对象，你可以：

```
使用 Rows.InsertNode 方法的返回值：
Var node = _flex.Rows.InsertNode(index, level);
```

或者，你可以用行的节点属性来在现有的一行检索节点：

```
Var node = _flex.Rows[index].IsNode
_flex.Rows[index].Node
 null;
```

无论哪种方式，一旦你有了一个“节点”对象，你就可以使用以下属性和方法来操作它：

- **等级**：在大纲树型图中获取或设置节点级别。
- 数据：在单元格中获取或设置由 Node.Row 和 Tree.Column 定义的值。
- 图片：在单元格中获取或设置由 Node.Row 和 Tree.Column 定义的图像。
- 选中：获取或设置由 Node.Row 和 Tree.Column 定义的单元格的选中状态。
- **折叠/扩展**：获取或设置节点的折叠/展开状态。

你还可以使用下列方法来探讨的大纲结构：

- **获取单元格区域**：获取**单元格区域**的对象，它描述了属于这个节点的行的范围。
- **子节点**：获取此节点下的子节点的数目。
- **节点**：获取一个节点的包含此节点的子节点数组。
- **获取节点**：获取与这个节点有一个给定关系的节点（父母，第一个孩子，下一个兄弟，等等）。

上面的讨论集中在绑定的情况下，即表格连接到提供数据的数据源的地方。你还可以创建绑定场景中的树型图和大纲。在这种情况下，事情其实比较简单，因为你可以通过将“**是节点**”属性设置为“真”来将任何一个节点变成一个节点行。

如果表格是未绑定的，它拥有所有显示的数据，并且当一个数据源拥有这些数据时，你做的是不可能的事情。例如，你可以像 C1FlexGrid 提供的“**树型图节点**”样品所显示的那样，使用 Move 方法来移动树型图周围的节点。

使用一个未绑定的表格中的节点与使用常规的“**树型视图**”控件的节点是非常相似的。

3.9 保存、加载和打印

C1FlexGrid 控件具有让你保存、加载并打印表格的方法。

3.9.1 保存和载入表格到文本文件

“保存表格”方法可以将表格内容保存到一个文本文件。该方法具有可以控制使用的分隔符类型的参数（例如，逗号、制表符、自定义分隔符），可以确定是否要保存固定的单元格还是只保存滚动的单元格，和要使用的编码类型（例如，ASCII 或 Unicode）。生成的文本文件随后可以被装入控件，或装入到其他支持逗号或制表符分隔的文件的应用程序（例如，Microsoft Excel）。

“加载表格”方法可以从文本文件中加载数据。你可以加载用“保存表格”方法或与其他应用程序创建的文本文件。文本文件的格式是相当简单的。单元格的内容可以保存为格式化字符串（完全是因为他们在屏幕上显示）。如果单元格的文本包含引号字符或单元格分隔符，那么单元格会被引号引起来。

单元格的文本中包含的任何引用字符都增加了一倍。这也是在 Microsoft Excel 文本文件中所使用的惯例。

文本文件不包含图片或格式信息。

“保存表格”方法有一个*标志*参数，它可以使你指定是否要保存整个表格或只保存其中的某些部分（滚动的、可见的、或选中的）。

3.9.2 保存和加载 Microsoft Excel 文件

从版本 2.5 开始，你可以使用“保存表格”和“加载表格”方法来保存和加载 Microsoft Excel 文件（.XLS）以及文本文件。这可以使你除了保存数据之外，还可以保存格式信息。要使用“保存表格”和“加载表格”方法来保存和加载 Excel 文件，只需简单地设置格式参数到 **FileFormatEnum.Excel**，并且像往常一样调用方法。你不需要在你的计算机上安装 Microsoft Excel。

Excel 文件中包含由“工作表”组成的“工作簿”。“保存表格”和“加载表格”方法总是用一个单一的工作表来保存工作簿，并从现有工作簿来载入第一页工作表。如果你想额外的控制工作来加载或保存，请使用 SaveExcel，LoadExcel，LoadExcelSheetNames 方法来代替。Excel 文件保存和加载的过程将转换大多数的数据类型和格式信息，包括行和列的尺寸、字体、颜色、格式、单元格对齐方式。然而，并不是所有的格式元素可以转换。例如，表格将加载 Excel 单元格的值，但它不会加载其隐含的公式。其他特征，如固定和合并单元格，图片，数据映射，单元格边框等，也没有被翻译。

3.9.3 从数据库中载入表格

你也可以从数据库中加载表格数据。这不同于数据绑定，它可以保持一个或多个控件和基础数据源之间的实时连接。为了从数据库加载数据，你可以使用 DataReader 对象，如下所示：

- Visual Basic

```
Private Sub _btnData_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles _btnData.Click
' 准备好 DataReader。
Dim strConn As String = "data source=MYMACHINE;initial
catalog=Northwind;"
Dim myConn As New SqlClient.SqlConnection(strConn)
Dim myCMD As New SqlClient.SqlCommand("SELECT * FROM Employees",
myConn)
myConn.Open()
Dim myReader As SqlClient.SqlDataReader = myCMD.ExecuteReader()
' 从 DB 模式来建立表格结构。
Dim dt As DataTable = myReader.GetSchemaTable()
_flex.Cols.Count = 1
Dim dr As DataRow
For Each dr In dt.Rows
Dim c As C1.Win.C1FlexGrid.Column = _flex.Cols.Add()
c.Caption =(c.Name <= CStr(dr("ColumnName")))
c.DataType = CType(dr("DataType"), Type)
Next dr
' 填充表格。
_flex.Rows.Count = 1
Dim row As Integer = 1
Dim cols As Integer = dt.Columns.Count
Dim v As Object() = CType(Array.CreateInstance(GetType(Object), cols),
Object())
While myReader.Read()
myReader.GetValues(v)
_flex.AddItem(v, row + 1, 1)
End While
' 清理。
_flex.AutoSizeCols()
myReader.Close()
myConn.Close()
```

- C#

```
private void _btnData_Click(object sender, System.EventArgs e)
{
// 准备好 DataReader。
string strConn = "data source=MYMACHINE;initial catalog=Northwind;";
System.Data.SqlClient.SqlConnection myConn = new
System.Data.SqlClient.SqlConnection(strConn);
System.Data.SqlClient.SqlCommand myCMD = new
System.Data.SqlClient.SqlCommand("SELECT * FROM Employees", myConn);
myConn.Open();
System.Data.SqlClient.SqlDataReader myReader = myCMD.ExecuteReader();
// 从 DB 模式来建立表格结构。
DataTable dt = myReader.GetSchemaTable();
_flex.Cols.Count = 1;
foreach (DataRow dr in dt.Rows)
{
Column c = _flex.Cols.Add();
c.Caption = c.Name = (string)dr["ColumnName"];
c.DataType = (Type)dr["DataType"];
// 填充表格。
_flex.Rows.Count = 1;
int row = 1;
int cols = dt.Columns.Count;
object[] v = (object[])Array.CreateInstance(typeof(object), cols);
while (myReader.Read())
{
myReader.GetValues(v);
_flex.AddItem(v, row++, 1);
}
// 清理。
_flex.AutoSizeCols();
myReader.Close();
myConn.Close();
}
```

3.9.4 打印表格

使用“打印表格”方法来打印表格的内容。该方法具有可以让你选择缩放模式，是否显示打印/预览对话框，设置页眉和页脚，等等的参数。

“打印参数”属性可以公开额外的打印性能，如字体，使用页眉和页脚，

而一个.NET Framework “**打印文档**”对象可以用来选择打印机，纸张大小和方向，页边距等。

下面的代码使用了“打印参数”属性来设置页面方向、页边距、页眉和页脚的字体。然后它调用“打印表格”方法来显示打印预览对话框窗口：

- Visual Basic

```
    ' 获取表格的“打印文档”对象。
    Dim pd As Printing.PrintDocument
    pd = _flex.PrintParameters.PrintDocument()
    '  设置页面（横向打印，左边页边距 1.5"）。
    With       pd.DefaultPageSettings
    .Landscape = True
    .Margins.Left = 150
    End With
    ' 设置页眉和页脚的字体。
    _flex.PrintParameters.HeaderFont = New Font("Arial   Black", 14,
    FontStyle.Bold)
    _flex.PrintParameters.FooterFont = New Font("Arial    Narrow",       8,
    FontStyle.Italic)
    ' 预览表格。
    _flex.PrintGrid("C1FlexGrid",
    C1.Win.C1FlexGrid.PrintGridFlags.FitToPageWidth +
    C1.Win.C1FlexGrid.PrintGridFlags.ShowPreviewDialog, "C1FlexGrid" + Chr(9)
+
```

- C#

```
// 获取表格的“打印文档”对象。
System.Drawing.Printing.PrintDocument pd =
_flex.PrintParameters.PrintDocument;
// 设置页面（横向打印，左边页边距 1.5"）。
pd.DefaultPageSettings.Landscape = true;
pd.DefaultPageSettings.Margins.Left = 150;
// 设置页眉和页脚的字体。
_flex.PrintParameters.HeaderFont = new Font("Arial Black",  14,
FontStyle.Bold);
_flex.PrintParameters.FooterFont = new Font("Arial Narrow",8,
FontStyle.Italic);
// 预览表格。
_flex.PrintGrid("C1FlexGrid", PrintGridFlags.FitToPageWidth |
PrintGridFlags.ShowPreviewDialog, "C1FlexGrid\t\t" +
Microsoft.VisualBasic.Strings.Format(DateTime.Now, "d"), "\t\tPage {0} of
{1}");
```

3.10 C1FlexGrid 过滤

表格中的数据过滤通常有两种形式：

- **基于表头**：过滤器的图标出现在有一个过滤器适用于它的每一列。用户可以通过点击过滤器的图标来查看和编辑过滤器。这是 Windows 7 或 Vista 或 C1FlexGrid 控件使用的机制。这种类型的过滤器的主要优点是：（1）用户可以看到哪些列被过滤了，（2）过滤不需要屏幕上的额外的不动产，（3）这种类型的过滤器可以更好地过滤编辑器并更容易定制。
- **过滤器行**：过滤器行保持始终可见，使用户可以直接到该行中键入值或表达式。这种类型的过滤器的主要优点是，用户随时都可以看到哪些列正在被过滤和当前过滤器的标准是什么。主要缺点是过滤器占用一些不动产，且可能会干扰常规的表格运行。虽然过滤器行没有建立在 C1FlexGrid 控件上，但他们实施起来还是相对容易。我们提供了一个“过滤器行”示例来显示如何做到这一点。

下面介绍的代码样本，主要取自包括产品的两个新样本：**列过滤器**和**自定义过滤器**。请参阅表明行动的特点的完整的项目样本。

3.10.1 允许过滤属性

要使用以表头为基础的过滤器，C1FlexGrid 控件遵循了与用来实现列的移动和按大小排序的相同的模式。表格有一个新的“允许过滤”属性，可用于控制在控件级别过滤，并且表格的列对象也有一个“允许过滤”属性，可用于控制在列级别过滤。

要启用简单的过滤方案，用户只需设置表格的“允许过滤”属性为“**真**”。然后，他们可以通过改变列的“允许过滤”属性的值来禁用或自定义过滤特定列的行为。列的“允许过滤”属性可以设置为下列值之一：

- **默认**：表格会自动创建一个 “列过滤”类型的过滤器。该过滤器将“值过滤”和“条件过滤”结合了起来，两者都在下文中有所描述。
- **根据值**：表格会自动创建一个 “值过滤”类型的过滤器。该过滤器包含一个应显示的值的列表。任何列表上不存在的值，最终用户都是看不到的。
- **根据条件**：表格会自动创建一个“条件过滤”类型的过滤器。该过滤器会指定两个条件，如“大于”或“包含”。这些条件可以与 AND 或 OR 运算符号结合起来。
- **自定义**：表格不会自动创建一个过滤器。开发人员预计实例化一个过滤器，并明确将其指定道列的“过滤器”属性。
- **无**：该列不能进行过滤。

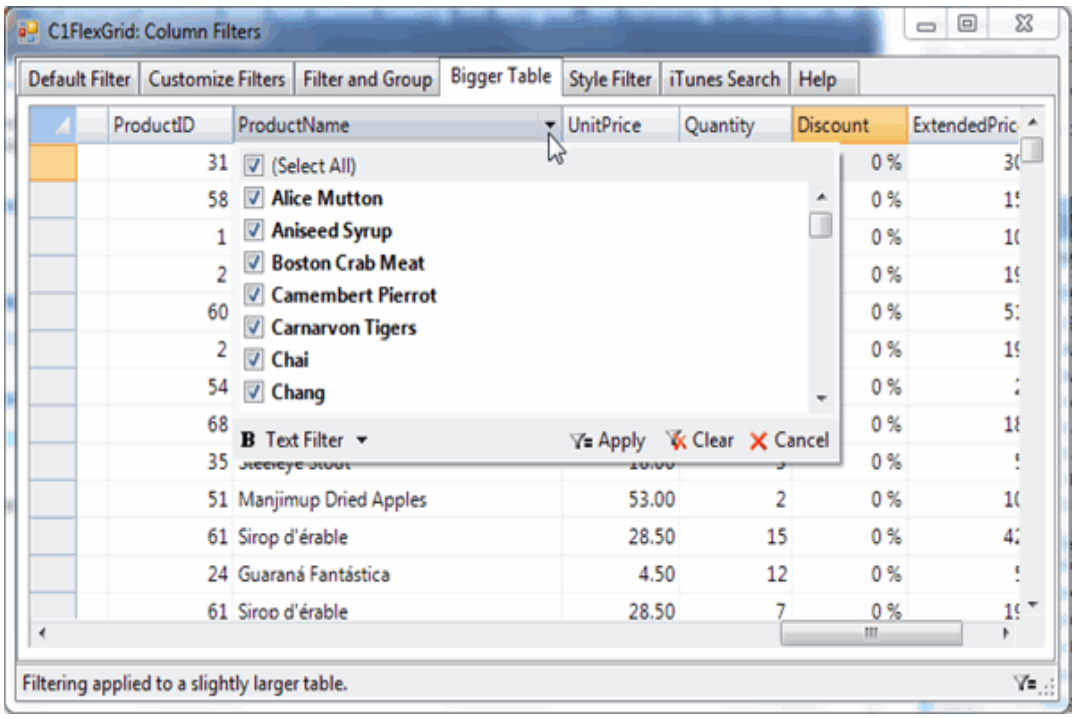
默认情况下，C1FlexGrid 控件会为使用指定的语言通过 **CurrentUICulture** 设置来将列过滤编辑器本地化。但是，你可以使用“语言”属性来推翻默认并指定当表格显示列过滤编辑器时应该使用的语言。

值过滤器

“值过滤器”概念上非常简单。它包含一个值的列表，并且只有该表中列出的值才会显示在表格上。如果列表设置为“无效的”，那么该过滤为“没有活性的”，且所有的值会被显示。这种类型的过滤器在过滤包含离散值，如名称或枚举，的列方面是被推荐的。

“值过滤”编辑器包括一个有复选框的值的列表。用户可以一次选中或取消选中所有值。采用先进的内置键盘导航来浏览长的列表是很容易的。编辑器中实现一个灵活的搜索缓冲区，可以使用户通过输入值的任何部分来找到值。例如，输入“希尔顿”，将选择下一个其中包含“希尔顿”的值，包括“纽约希尔顿”，“王子爱德华希尔顿”，或“巴黎希尔顿”。此外，敲 Ctrl+向上键或 Ctrl +向下键将会导航到下一个或前一个检查项目。使用目前分配给列的格式，值就会显示在列表上。

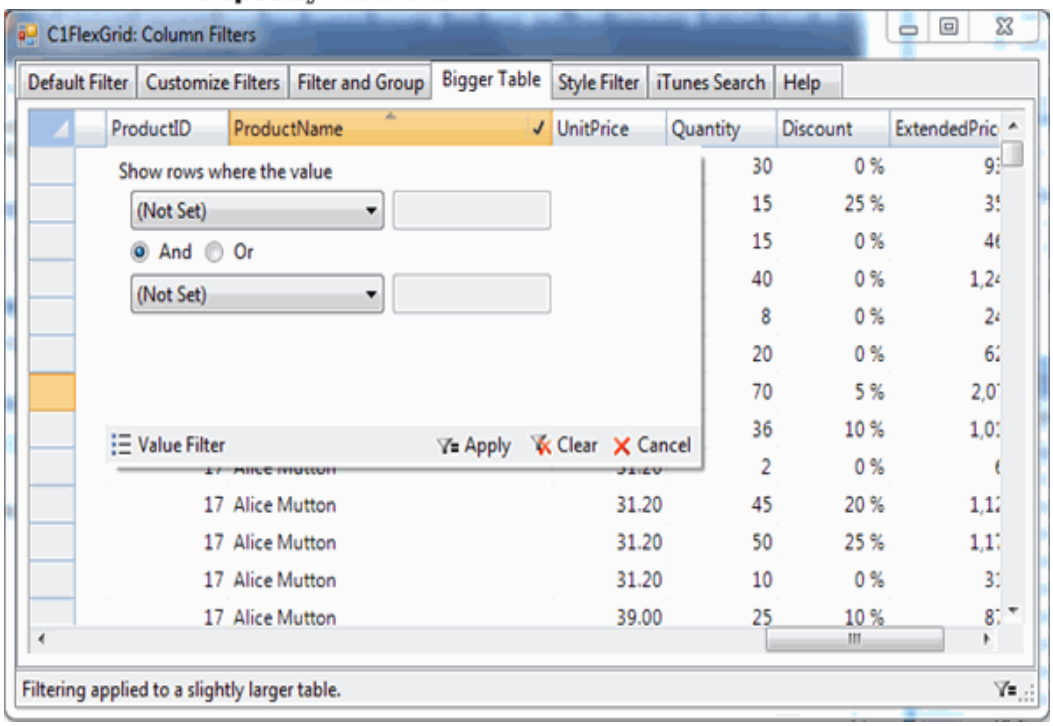
这个图像显示了“值过滤”编辑器。



条件过滤器

“条件过滤器”比其他的更加灵活。它不是选择特定的值，而是允许用户用运算符号来指定两个条件，如“大于”、“开始”或“包含”。这种类型的过滤器在过滤包含“连续的”值的列，如数字，或日期/时间值，在这方面是被推荐的。

这个图像显示了“条件过滤”编辑器。



有过滤器适用于列来显示他们的标题过滤器的图标，甚至当鼠标不在他们上面的时候。在这个图像中你可以看到有“**产品名称**”和“**数量**”列标题的地方显示过滤器的图标。

内置的过滤器支持自动定位在以下语言：英语，西班牙语，法语，意大利语，葡萄牙语，德国，荷兰，俄罗斯，日本，希腊，丹麦，芬兰，挪威，瑞典文，阿拉伯文，波兰，中国，土耳其，波斯语，朝鲜语，希伯来语。该本地化资源是内置的，并不需要提供额外的 DLL

自定义过滤器

可以创建自定义过滤器来处理专门的值。例如，自定义过滤器在过滤颜色、地理或自定义数据类型方面值得推荐。

要创建一个自定义的过滤器，开发人员必须创建两个类：

- **过滤器**：这个类必须实现 IC1ColumnFilter 接口，它可以指定过滤器应用到一个特定的值，对过滤器进行复位，并返回一个用于查看和编辑过滤器的参数的编辑器。
- **过滤编辑器**：这个类必须继承自 Control，必须实现 IC1ColumnFilterEditor 的接口，该接口可以指定用于初始化编辑器和更改应用到过滤器的方法。

自定义过滤器的样本包含三个自定义过滤器，用于过滤类型的**颜色**，**日期/时间**和**字符串**的值。

3.10.2 程序化地管理过滤器

正如我们前面提到的这个文件，设置表格的“允许过滤”属性为“**真**”，这足以让所有的列进行列过滤。然而，在许多情况下，你可能需要更精细的过

滤控制。这可以通过修改个别列的“允许过滤”和“过滤”属性来实现。例如，下面的代码使能够启用过滤，但对过滤**字符串**类型的列进行了限制：

```
// 绑定和配置表格
flex.DataSource = dtProducts;
_flex.Cols["UnitPrice"].Format = "#,###.00";
// 启用过滤
_flex.AllowFiltering = true;
// 限制过滤 “字符串” 类型的列
foreach (Column c in _flex.Cols)
{
c.AllowFiltering = c.DataType == typeof(string)
? AllowFiltering.Default
: AllowFiltering.None;
}
```

你可以通过创建过滤器并将他们分配到列，或通过检索现有的过滤器并修改其属性，来进一步自定义过滤过程。例如，下面的代码创建了一个“条件过滤器”，配置它来选择所有以字母“C”开头的项目，然后分配这个新的过滤器给“产品名称”一列：

```
// 创建一个新的 “条件过滤器”
var filter = new ConditionFilter();
// 配置过滤器来选择以“C” 开始的项目
filter.Condition1.Operator = ConditionOperator.BeginsWith;
filter.Condition1.Parameter = "C";
// 分配新的过滤器到 “产品名称”列
_flexCustom.Cols["ProductName"].Filter = filter;
```

3.10.3 程序化地应用过滤器

当用户编辑他们或当他们适用于一列时，过滤器是适用的。当数据发生变化时，它们不会自动应用。

要将过滤器应用到从表格加载的当前的数据，请调用表格的“应用过滤器”方法。

例如，当用户编辑表格上的数据时，下面的代码启用了一个“**应用过滤器**”按钮。点击这个“**应用过滤器**”按钮即可应用该过滤器，并可以直到下一次的变化前禁用该按钮。

```
public Form1()
{
InitializeComponent();
// 获取一些数据
var da = new OleDbDataAdapter("select * from products",
GetConnectionString());
var dtProducts = new DataTable();
da.Fill(dtProducts);
// 将表格绑定到数据_
flex.DataSource = dtProducts;
// 启用过滤
_flex.AllowFiltering = true;
// 监测变化以便启用“应用过滤器”按钮
_flex.AfterEdit += _flex_AfterEdit;
}
```

上面的代码可以将一个表格绑定到数据源，可以通过将“允许过滤”属性设置为“真”来启用过滤器，并可以连接一个事件处理程序到“**编辑后**”事件。事件处理程序的执行情况如下：

```
void _flex_AfterEdit(object sender, C1.Win.C1FlexGrid.RowColEventArgs
e)
{
foreach (C1.Win.C1FlexGrid.Column c in _flex.Cols)
{
if (c.ActiveFilter != null)
{
_btnApplyFilters.Enabled = true;
break;
}
}
}
```

此代码可以扫描所有列，以确定一个过滤器是否为任何列所定义。如果检测到有一个正在起作用的过滤器，该代码可以启用将过滤器应用于目前的数据的按钮。当单击该按钮时，下面的事件处理程序会执行如下：

```
private void _btnApplyFilters_Click(object sender, EventArgs e)
{
_flex.ApplyFilters();
_btnApplyFilters.Enabled = false;
}
```

该代码简单适用于所有活跃着的过滤器，并且直到下一次的变化前才禁用按钮。相反，如果你不需要一个按钮，而只是简单地想在每次编辑后应用该过滤器，你可以从“**编辑后**”事件处理器那里直接调用“**应用过滤器**”，如下所示：

```
void _flex_AfterEdit(object sender, C1.Win.C1FlexGrid.RowColEventArgs
e)
{
_flex.ApplyFilters();
```

3.10.4 自定义过滤器的行为

当过滤器被应用后，表格会通过将“可见”属性设置为“**假**”来隐藏一些行。但表格也能够激发可以允许你自定义过滤行为的“过滤前”和“过滤后”事件。例如，假设，你不是要显示和隐藏行，而是要应用不同的风格来表明这些行是否能通过过滤器。这可以使用此代码来轻松地实现：

```
public Form1()
{
InitializeComponent();
// 配置表格_
flex.AllowFiltering = true;
_flex.DataSource = dtInvoices;
// 为过滤器排除的行创建样式
var cs = _flexStyles.Styles.Add("filteredOut");
cs.BackColor = Color.LightSalmon;
cs.ForeColor = Color.DarkOrange;
// 连接过滤前和过滤后事件的处理程序
_flex.BeforeFilter += _flex_BeforeFilter;
_flex.AfterFilter += _flex_AfterFilter;
}
```

该代码可以创建一个将被用于显示未通过过滤器的行（而不是使他们不可见）的自定义样式。其次，该代码可以将处理程序与“过滤前”和“过滤后”事件联系起来。事件处理程序列举如下：

在我们完成将自定义样式应用于被过滤掉的行之前，“过滤前”事件处理程序可以调用“开始更新”方法来防止表格自己进行重新绘制。“开始更新”和“结束更新”方法可以代替已被弃用的“重绘”属性。

“过滤后”事件处理程序可以通过检索我们创建的样式来开始显示已筛选

```
// 在应用过滤器之前暂停绘制
void _flex_BeforeFilter(object sender, CancelEventArgs e)
{
_flexStyles.BeginUpdate();
}
// 在应用过滤器之后应用样式
void _flexStyles_AfterFilter(object sender, EventArgs e)
{
// 获取用于显示筛选出的行的样式
var cs = _flex.Styles["filteredOut"];
// 将样式应用于所有行
for (int r = _flexStyles.Rows.Fixed; r < _flexStyles.Rows.Count;
r++)
{
var row = _flexStyles.Rows[r];
if (row.Visible)
{
// normal row, reset style
row.Style = null;
}
else
{
// 过滤行，使其可见并应用样式
row.Visible = true;
row.Style = cs;
}
}
// 恢复更新_
flexStyles.EndUpdate();
}
```

出的行。然后它可以对表格中的行进行扫描，并且将新的样式应用于“可见”属性设置为“**假**”的所有行。这些是被过滤器隐藏起来的行。一旦做到这一点，代码可以调用“结束更新”来恢复表格的更新。

3.10.5 自定义 UI 过滤

我们相信，默认的过滤行为和 UI 可以解决牵涉到列过滤中的绝大多数情况。但是，你可以独立地使用这些列过滤器列类，来实现自己的自定义用户界面。

例如，下面的代码显示了你可以如何使用“条件过滤器”的类来实现 C1FlexGrid iTunes 风格的搜索框。

这种类型的搜索，可以允许用户键入一个值，并自动过滤表格的行来显示任何列中包含搜索字符串的行。为落实 iTunes 风格的搜索，我们以一个包含将作为一个过滤器参数的文本的文本框开始。我们还定义了一个定时器，它将在用户停止往文本框中输入的几毫秒后适用于过滤器：

```
public Form1()
{
InitializeComponent();
// 在用户停止在搜索框中键入的 1/2 秒后配置定时器来适用于过滤器
_timer.Interval = 500;
_timer.Tick += t_Tick;
// 监测搜索框的变化
_txtSearch.TextChanged += _txtSearch_TextChanged;
}
// 当搜索框中的文字发生变化时重新启动定时器
void _txtSearch_TextChanged(object sender, EventArgs e)
{
_timer.Stop();
_timer.Start();
}
```

既然定时器已经被配置好了，所有我们需要做的就是在计时器滴答计时时来创建和应用过滤器：

```
// 在计时器滴答计时时应用过滤器
void t_Tick(object sender, EventArgs e)
{
// done for now...
_timer.Stop();
// 配置过滤器
var filter = new C1.Win.C1FlexGrid.ConditionFilter();
filter.Condition1.Operator =
C1.Win.C1FlexGrid.ConditionOperator.Contains;
filter.Condition1.Parameter = _txtSearch.Text;
// 应用过滤器
_flex.BeginUpdate();
for (int r = _flex.Rows.Fixed; r < _flex.Rows.Count; r++)
{
bool visible = false;
for (int c = _flex.Cols.Fixed; c < _flex.Cols.Count; c++)
{
if (filter.Apply(_flex[r, c]))
{
visible = true;
break;
}
}
_flex.Rows[r].Visible = visible;
}
_flex.EndUpdate();
}
```

3.11 C1FlexGrid 的属性组

C1FlexGrid 控件拥有一套非常丰富的属性，方法和事件。但你却无需为了有效地使用该控件，而知道所有的这些。

下面的参考资料显示了其中最为重要的属性，方法，以及按使用类型来分组的事件。而且其中某些元素不只出现在一个分组中。想知道有关于此的更多详细信息，请选择下面的特定元素。

表格布局

Rows, Cols, AutoSizeCols, ScrollBars

光标和选择

SelectionMode, Select, ShowCell, Row, Col, RowSel, ColSel, MouseRow, MouseCol,

BeforeRowColChange, AfterRowColChange, BeforeSelChange, AfterSelChange, KeyActionTab, KeyActionEnter

编辑

AllowEditing, ComboList, EditMask, BeforeEdit, StartEdit, ValidateEdit, AfterEdit, StartEditing, FinishEditing, Editor, CellButtonClick, KeyDownEdit, KeyPressEdit, KeyUpEdit, ChangeEdit

获取和设置值

Item (indexer), GetData, GetDataDisplay, SetData, GetCellRange, GetCellImage, SetCellImage, Clip, FindRow, Aggregate, CellChanged

用户界面

AllowEditing, AllowMerging, AllowResizing, AllowDragging, AllowSorting, BeforeSort, AfterSort, AutoSearch, AutoSearchDelay, BeforeDragColumn, AfterDragColumn, BeforeDragRow, AfterDragRow, BeforeResizeColumn, AfterResizeColumn, BeforeResizeRow, AfterResizeRow, ShowScrollTip

概述和总结

Subtotal, Tree, IsNode, Level, Collapsed, BeforeCollapse, AfterCollapse

合并单元格

AllowMerging

数据绑定

DataSource, DataMember, AfterDataRefresh, AutoResize, GridError

保存、加载和打印表格

LoadGrid, SaveGrid, LoadExcel, SaveExcel, ClipSeparators, PrintGrid

OLE 拖放

DragMode, DropMode, BeforeMouseDown

4. 数据绑定

数据绑定可以使一个或多个数据消费者以一种同步的方式被连接到一个数据提供商。如果你在一个数据绑定的表格上移动光标，那么连接到同一数据源的其他控件将会发生改变，以反映当前的新纪录。如果你在一个数据绑定的表格上编辑一个值，那么其他连接到同一数据源的控件将会发生改变，以反映新的值。

C1FlexGrid 控件支持将数据绑定到 ADO.NET 数据源对象，例如，**数据表** DataTable、**数据视图** DataView、**数据集** DataSet 和**数据视图管理器** DataViewManager。C1FlexGrid 也支持将数据绑定到 **ComponentOne DataObjects for WinForms** 组件，例如，C1Express 表、C1Express 视图、C1Express 连接、**C1 数据视图** C1DataView、**C1 数据表源** C1DataTableSource 和 **C1 数据集** C1DataSet。

要将表格绑定到一个数据源，必须将数据源对象分配到表格的“数据源”DataSource 属性。如果数据源对象包含多于一个的表，你还必须将“数据成员”DataMember 属性设置为一个能指定应该使用哪个表地字符串。

另外，你也可以用一个单一的 “设置数据绑定”SetDataBinding 方法的调用来同时指定两个属性。当你将一个新的数据源分配到表格，它将会将自己的列自动刷新来绑定数据源中可用的列。然后，你可以通过移动、隐藏或删除它们来自定义这些列。你还可以设置列的属性，如它们的宽度、编辑掩码和格式。

可用的示例项目

有关一个在绑定到数据源后将表格列重新排序的例子的详细信息，请在 ComponentOne 帮助中心参阅“**列顺序**”示例。

有关如何创建 ADO.NET 数据源对象的更多相关详细信息，请参阅.NET Framework 文档。

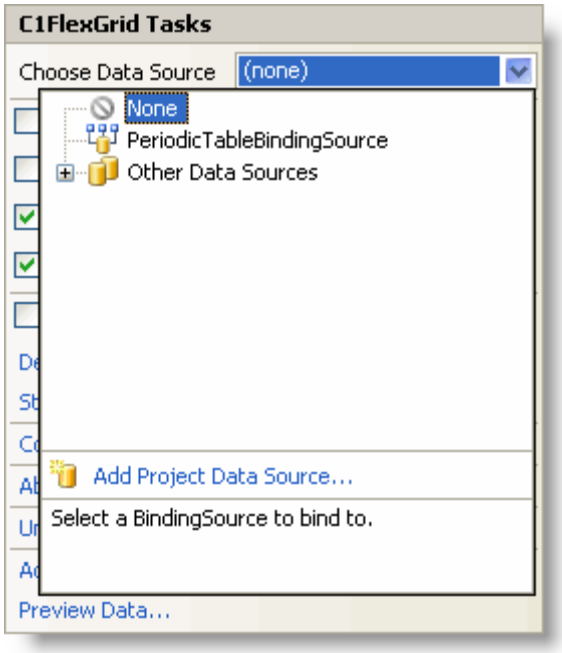
有关如何在 WinForms 中使用 ComponentOne 数据对象的更多相关详细信息，请参阅包含在 ComponentOne Studio Enterprise 和 ComponentOne Studio for WinForms 中的 ComponentOne DataObjects for WinForms 文档。

4.1 绑定到数据源

无需编写一行代码，你可以通过在 Visual Studio 中使用**数据源配置向导**，很容易地将 C1FlexGrid 绑定到一个数据源。要访问该向导有以下两种途径，可以通过在属性窗口中选择“**数据源**”属性，或通过 **C1FlexGrid 任务菜单**中的“**选择数据源**”对话框。有关 C1FlexGrid 任务菜单的更多详细信息，请参阅

“C1FlexGrid 任务菜单”（第 34 页）。

点击属性窗口中的“数据源”属性旁边的下拉箭头，或 C1FlexGrid 任务菜单上的“选择数据源” 对话框，可以允许你从一个可用数据源列表中选择，或者添加一个数据源到你的项目。要想添加一个数据源到你的项目，请单击“**添加项目数据源**”来打开**数据源配置向导**。

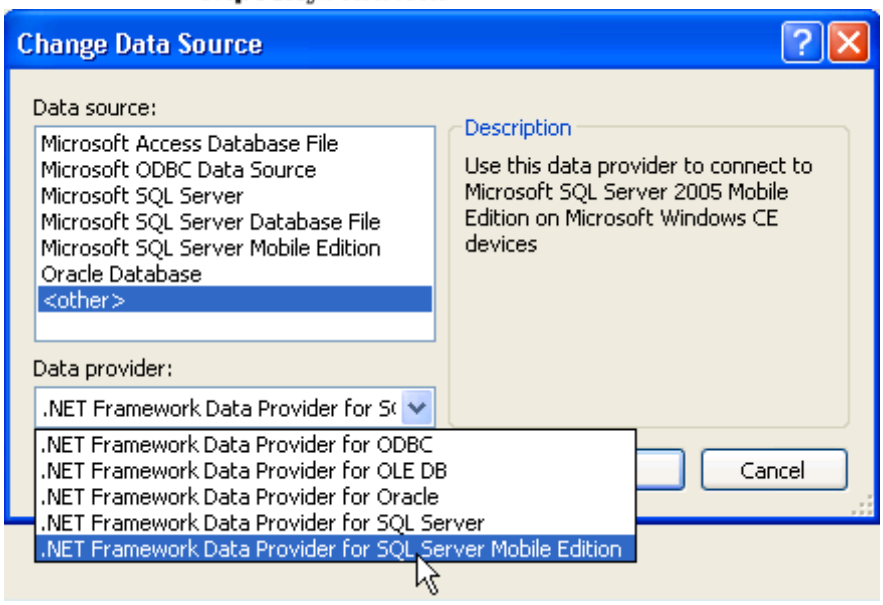


数据源配置向导可以引导你一步步地添加数据源。“**选择数据源类型**”页可以使你选择你所想要用来收集信息的应用程序的类型。默认的选择是数据库。

在“**选择你的数据连接**“页上，你可以指定数据库的位置。如果你尚未连接到数据库的话，你可以通过点击“**新建连接**”按钮来指定一个新的连接。点击“**新建连接**”按钮，打开“**添加连接对话框**，在这里你可以浏览到你的数据库的位置，并测试连接。

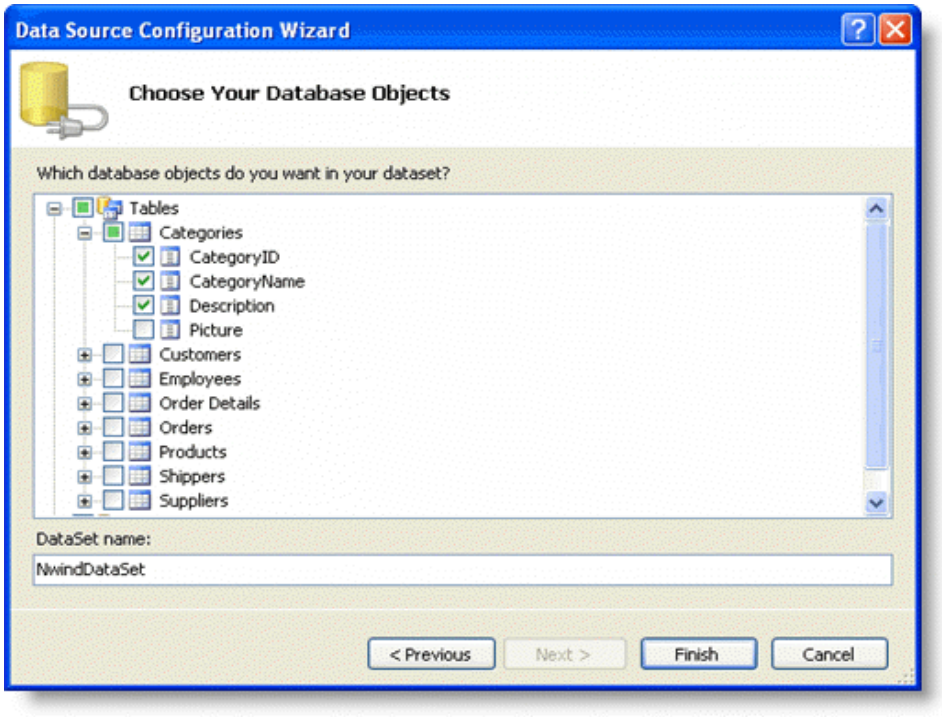
移动设备用户请点击这里以获得更多信息。

如果你正在使用 **WinForms** 的 **ComponentOne FlexGrid** 的移动版，请在“**添加连接**”对话框中，将数据源更改为一个由 Compact Framework 支持的，如 SQL Server 移动版的.NET Framework 数据提供者（.NET Framework Data Provider for SQL Server Mobile Edition）。



然后，你可以使用支持的数据库，例如，一个 SQL Server 移动版数据库。微软提供了一个 Northwind.sdf，默认情况下，它位于 C:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0 folder for Microsoft Visual Studio 2005。

该向导可以保存和命名你在“**保存连接字符串到应用程序配置文件**“页面上的连接。在此页面上，你可以指定一个名称或使用该向导提供的默认。在“**选择数据库对象**”页面上，你可以指定你要在“数据集”中包括的表和字段。你也可以为你的“数据集”提供一个名称或使用向导提供的默认。



该向导可以创建数据集、绑定源和表适配器，并且将它们添加到你的项目。请再次单击“**数据源**”属性旁边的下拉箭头或 C1FlexGrid 任务菜单上的“选择数据源”对话框，然后选择数据源。如果你双击这个窗体，你还会发现，“窗体_加载” Form_Load 事件自动生成了用来填充数据库的代码。

4.2 存储和检索数据

C1FlexGrid 控件，可用于绑定模式或未绑定模式。在绑定模式下，该表格连接到一个数据源，并且，来源于数据源的所有数据会显示在表格中。在这种模式下，改变表格上的数据就可以在基础数据源改变它。在未绑定模式下，表格自己管理自己的数据源。

不论是在绑定模式下或未绑定模式下，要访问 C1FlexGrid 中的数据，最简单的方法是使用行和列的索引。该索引允许你在从中可以获取或设置存储在那里的数据的行或列中指定一个单元格。

例如，下面的代码可以选择一行的第二个单元格中的数据：

- Visual Basic

```
Row(2).Selected = True
```

- C#

```
Row[2].Selected = true;
```

“项目” Item 属性是另一种简单的方法来访问 C1FlexGrid 中的数据。“项目” Item 属性是一个索引，它可以为行和列编纂一个索引，并获取或设置单元格中所存储的数据。（你还可以使用列名称作索引）。

例如，下面的代码可以将行数字存储在第一个表格列：

- Visual Basic

```
Dim r As Integer
For r = _flex.Rows.Fixed To _flex.Rows.Count - 1
_flex(r, 0) = r
Next
```

- C#

```
for (int r = _flex.Rows.Fixed; r <= _flex.Rows.Count - 1; r++)
{
_flex[r, 0] = r;
}
```

当你指定一个值到一个单元格，该表格会试图将这个值转换成列的指定的数据类型。如果该转换失败的话，表格会触发“表格错误”GridError 事件，但并不会改变单元格。你可以通过使用“设置数据”SetData 方法来重写此行为，并且将“强制”*coerce* 参数设置为“**假**”。

当你使用索引检索数据的时候，表格可以使单元格中存储的实际数据返回。要想检索一个包含数据格式化版本的字符串（表格向用户显示的内容），请使用“获取数据显示”GetDataDisplay 方法。你还可以通过使用 Clip 属性来设置和检索所选中的内容。此属性在处理剪贴板和拖放操作方面是特别有用的。在默认情况下，Clip 属性可以使一个包含制表符（CHR（9））的字符串返回到单元格与单元格之间，并且使含有回车符（CHR（13））返回到行与行之间。要想使用不同的分隔符，请改变“片段分离”ClipSeparators 属性。

最后，你可以通过使用“单元格区域”对象（第 45 页）来设置和检索任意单元格区域内的内容。

5. FlexGrid for WinForms 示例

请注意，ComponentOne 软件工具伴随着各种示例项目和/或演示，这可能会使用到包括 ComponentOne Studios 的其他开发工具。

可通过 **ComponentOne Sample Explorer** 来访问示例。要查看示例的话，请在桌面上单击“**开始**”按钮，然后单击“**所有程序**” All Programs | Studio for WinForms | Samples **示例**| FlexGrid。

- Visual Basic 示例

示例	描述
自动调整尺寸编辑 AutoSizeEdit	该示例演示了应该如何在用户输入的时候调整编辑器。该示例使用了 C1FlexGrid 控件。
条形图表 BarChart	该示例使用了 OwnerDrawn 单元格来建立起基于表格数据的图表。该示例使用了 C1FlexGrid 控件。
信号灯	该示例演示了应该如何使用 CellStyles“单元格样式”来使单元格闪烁。该示例可以创建一 Blinker 这将导致所有的单元格具有这种闪烁的样式。该示例使用了 C1FlexGrid 控件。
绑定模式下删除 BoundDelete	该示例可以从绑定的 FlexGrid 删除行。该示例使用了 C1FlexGrid 控件。
绑定模式下完成编辑 BoundFinishEdit	该示例演示了应该如何在编辑后向“数据行” DataRow 提交更改。该示例使用了 C1FlexGrid 控件。
绑定模式下的图像	该示例演示了应该如何将表格绑定到作为 OLE 对象存储的图像领域。该示例使用了 BoundImageC1FlexGrid 控件。
C1FlexGrid 可打印 C1FlexGridPrintable	该示例提供了一个可打印的（通过 C1Preview）C1FlexGrid 控件。该示例使用了 C1FlexGrid 控件，并调用了 C1.C1Preview 命名空间。
单元格边框 CellBorders	该示例使用了“所有者绘制” OwnerDraw 来提供自定义的单元格边框。该示例使用了 C1FlexGrid 控件。

单元格数据样式 CellDataType	该示例在每一个单元格的基础上分配单元格类型（和编辑器）。该示例使用了 C1FlexGrid 控件。
单元格合并 CellMerging	该示例在 C1FlexGrid 控件中显示了不同类型的单元格合并。该示例使用了 C1FlexGrid 控件。
单元格注释 CellNotes	该示例可以将注释附加到 C1FlexGrid 的单元格上，并且可以通过以下两大类：单元格注释 CellNote 和单元格注释管理器 CellNoteManager 来实现 Excel 样式的单元格注释。
列编辑器 ColumnEditor	该示例可以在你的控件中显示 C1FlexGrid 设计时列编辑器。该示例使用了 C1FlexGrid 控件。
自定义数据映射 CustomDataMap	该示例演示了应该如何在使用“数据映射” DataMaps 时自定义下拉列表选项。该示例使用了 C1FlexGrid 控件。
自定义编辑器 CustomEditor	该示例演示了你应该如何配置自己的单元格编辑器。该示例使用了 C1FlexGrid 控件。
自定义合并 CustomMerge	该示例演示了你应该如何实现用自己的合并逻辑来创造一个电视指南显示。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 2 CustomMerge2	该示例演示了应该如何实现自己的合并逻辑并由此将合并模式混合到一起。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 3 CustomMerge3	该示例演示了应该如何通过重写“获取数据” GetData 方法来自定义分组。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 4 CustomMerge4	该示例演示了应该如何通过重写“获取数据” GetData 方法来自定义分组。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义打印 CustomPrint	该示例可以使用“创建图像” createImage 方法来打印包含任意行和列碎片的表格。该示

	例使用了 C1FlexGrid 控件。
自定义树型图 CustomTree	该示例可以使用 C1FlexGrid 控件来创建一个树型图。该示例使用了 C1FlexGrid 控件。
数据索引 DataIndex	该示例可以使用 Row.DataIndex 属性来获取基础数据行。该示例使用了 C1FlexGrid 控件。
数据映射 DataMap	该示例可以在绑定到数据源时使用数据映射列。该示例使用了 C1FlexGrid 控件。
数据读取器 DataReader	该示例可以从一个数据读取器 DataReader 中填充表格。该示例使用了 C1FlexGrid 控件。
数据表 DataTable	该示例演示了应该如何创建、填充并绑定一个“数据表” DataTable 对象到表格（包括添加或移除行）。该示例使用了 C1FlexGrid 控件
数据树型图 DataTree	该示例可以将表格绑定到一个分层的数据源，并且在子表格中显示详情。
数据库动态样式 DBDynamicStyles	该示例可以根据其内容来指定样式到表格的单元格。该示例使用了 C1FlexGrid 控件。
数据库图像区域 DBImageField	该示例可以显示“数据表” DataTable 中存储的图像。该示例使用了 C1FlexGrid 控件。
数据库架构更改 DBSchemaChange	该示例演示了应该如何测试表格对数据源对象发生变化的响应。该示例使用了 C1FlexGrid 控件。
数据库树型图 DBTree	该示例说明了应该如何在一个分层树型视图中显示绑定的数据。该示例使用了 C1FlexGrid 控件。
拖放 DragDrop	该示例演示了应该如何自定义自动拖放。该示例使用了 C1FlexGrid 控件。
手动拖放 DragDropManual	该示例演示了应该如何在控件之间与表格内部实现手动的“OLE 拖放” OleDragDrop。

拖动行 DragRow	该示例演示了应该如何在表格之间拖动行。该示例使用了 C1FlexGrid 控件。
动态样式 DynamicStyles	该示例演示了应该如何指定样式到一个以另一个单元格的内容为基础的单元格。该示例使用了 C1FlexGrid 控件。
编辑 Editing	该示例演示了应该如何使用文本框、列表、掩码和复选框来对单元格进行编辑。该示例使用了 C1FlexGrid 控件。
含总计的过滤行 FilterRow_With_Totals	该示例显示了应该如何添加总计到一个含有“过滤行” FilterRow 的绑定表格。
FlexByRow	该示例设置了一个“换位”格式的表格。在该示例中，通常被分配到表格列的数据类型、编辑掩码、格式和其他属性被分配到了表格行。该示例使用了 C1FlexGrid 控件。
FlexGroup	该示例演示了你应该如何通过使用 C1FlexGrid 来实现 Outlook 风格的分组和过滤。该示例使用了 C1FlexGrid 控件。
固定底部 FreezeBottom	该示例演示了应该如何在表格的底部显示固定的行。该示例使用了 C1FlexGrid 控件。
GridChart	该示例演示了你可以如何将一个 FlexGrid 附加到一个 C1Chart 控件。该示例使用了 C1FlexGrid 和 C1Chart 控件。
HierData	该示例演示了你应该如何绑定到分层数据源（主从风格）。该示例使用了 C1FlexGrid 控件。
超链接 Hyperlink	该示例演示了你应该如何添加超链接到一个 C1FlexGrid 中的单元格。该示例使用了 C1FlexGrid 控件。
加载速度 LoadSpeed	该示例显示了加载数据到 C1FlexGrid 的三种技术。
映射和分组	该示例显示了在使用数据映射列时的分组和排序行为。该示例使用了 C1FlexGrid 控件。

合并样式 MergeStyles	该示例合并了分配到行、列和单元格的单元格样式 CellStyles。该示例使用了 C1FlexGrid 控件。
多列词典 MultiColumnDictionary	该示例使用了“多列词典”MultiColumnDictionary 类来实现多列下拉菜单。
大纲 Outline	该示例演示了应该如何隐藏大纲中的重复数据。该示例使用了 C1FlexGrid 控件。
PdfExportVB	该示例使用了 C1Pdf 方法来将 C1FlexGrids 导出为 PDF 文件。
产品结构树 ProductTree	该示例演示了应该如何建立一个关于产品信息的自定义结构树。该示例使用了 C1FlexGrid 控件。
RTF 表格 RTFGrid	该示例演示了应该如何在表格的单元格中显示 RTF 文本。该示例使用了 C1FlexGrid 控件。
滚动位置 ScrollPosition	该示例演示了“滚动位置”ScrollPosition 属性是如何运作的。该示例使用了 C1FlexGrid 控件。
选择模式 SelectionMode	该示例显示了“选择模式”SelectionMode 属性的效果。该示例使用了 C1FlexGrid 控件。
排序 Sorting	该示例显示了一些排序技术。该示例使用了 C1FlexGrid 控件。
拆分 Splits	该示例演示了应该如何将一个 C1FlexGrid 拆分成多个视图。该示例使用了 C1FlexGrid 控件。
开始编辑 StartEditing	该示例实现了一个停留在单元格编辑模式的表格。该示例使用了 C1FlexGrid 控件。
样式 Styles	该示例演示了应该如何使用“样式”来自定义 C1FlexGrid 的外观。该示例使用了 C1FlexGrid 控件。

分类汇总 Subtotals	该示例可以创建多个列的分类汇总。该示例使用了 C1FlexGrid 控件。
移调 Transpose	该示例演示了应该如何在一个表格中移调数据。该示例使用了 C1FlexGrid 控件。
TreeCheck	该示例演示了应该如何添加复选框到一个表格的树型图。该示例使用了 C1FlexGrid 控件。
树型图节点 TreeNode	该示例演示了应该如何使用 C1FlexGrid 的 Node 对象来管理大纲树型图。该示例使用了 C1FlexGrid 控件。
验证 Validate	该示例演示了应该如何与 C1FlexGrid 控件一起使用“错误提供商” ErrorProvider 控件。该示例使用了 C1FlexGrid 控件。
变焦单元格 ZoomCell	该示例演示了应该如何放大选定的单元格。

- C# 示例

示例	描述
加速器说明 AcceleratorCaption	该示例演示了应该如何将快捷键添加到表格的标题。
分析 Analyze	该示例演示了应该如何提供动态数据的排序和分组。该示例使用了 C1FlexGrid 控件。
动画 GIF AnimagedGif	该示例说明了应该如何在表格的单元格中显示 GIF 动画。
自动完成 AutoComplete	该示例演示了应该如何在用户输入时完成登录。
自动调整尺寸编辑 AutoSizeEdit	该示例演示了应该如何在用户输入时调整编辑器。该示例使用了 C1FlexGrid 控件。
条形图表 BarChart	该示例演示了应该如何使用表格中的单元格来绘制条形图表。该示例使用了 C1FlexGrid 控件。

信号灯 Blinker	该示例演示了应该如何使用“单元格样式”CellStyles 来使单元格闪烁。
绑定模式下删除 BoundDelete	该示例演示了应该如何从一个绑定的 FlexGrid 中删除行。该示例使用了 C1FlexGrid 控件。
绑定模式下完成编辑 BoundFinishEdit	该示例演示了应该如何在编辑后提交更改到一个“数据行”DataRow。该示例使用了 C1FlexGrid 控件。
绑定模式下的图像映射 BoundImageMap	该示例演示了应该如何通过一个数据绑定的表格来使用“图像映射”ImageMap 属性。该示例使用了 C1FlexGrid 控件。
C1FlexGrid 可打印 C1FlexGridPrintable	该示例提供了一个可打印的（通过 C1Preview）C1FlexGrid 控件。该示例使用了 C1FlexGrid 控件，并且调用了 C1.C1Preview 命名空间。
单元格边框 CellBorders	该示例演示了应该如何使用“所有者绘制”OwnerDraw 来提供自定义的单元格边框。该示例使用了 C1FlexGrid 控件。
单元格的数据类型 CellDataType	该示例演示了应该如何在每个单元格的基础上分配单元格样式（和编辑器）。该示例使用了 C1FlexGrid 控件。
单元格标签延迟 CellLabelDelay	该示例演示了应该如何在指定的时间间隔后显示显示单元格标签。
单元格注释 CellNotes	该示例可以将注释附加到 C1FlexGrid 的单元格上，并且可以通过以下两大类：单元格注释 CellNote 和单元格注释管理器 CellNoteManager 来实现 Excel 样式的单元格注释。
经典 Classic (C1FlexGridClassic)	该示例演示了应该如何使用 C1FlexGrid 2010 的过滤功能。该示例演示了你应该如何通过使用“允许过滤”AllowFiltering 属性和“过滤后”AfterFilter 事件来控制和自定义表格的内置过滤器。

列顺序 ColumnOrder	该示例演示了应该如何动态地定位列的位置。该示例使用了 C1FlexGrid 控件。
列宽提示 ColumnWidthTip	该示例演示了应该如何在用户调整列宽时显示一个“工具提示”ToolTip。
交叉表 CrossTabs	该示例演示了应该如何使用数据透视表来总结和交叉引用数据。
自定义数据 CustomData	该示例演示了应该如何实现一个自定义的数据源对象。该示例使用了 C1FlexGrid 控件。
自定义数据映射 CustomDataMap	该示例演示了应该如何在使用“数据映射”DataMaps 时自定义下拉列表中的选项。该示例使用了 C1FlexGrid 控件。
自定义编辑器 CustomEditor	该示例演示了应该如何实现自己的单元格编辑器。该示例使用了 C1FlexGrid 控件。
自定义编辑器 CustomEditors	该示例演示了应该如何使用内置.NET 和用 C1FlexGrid 来自定义编辑控件。
自定义过滤器 CustomFilters	该示例演示了应该如何实现 C1FlexGrid 的自定义过滤器。
自定义合并 CustomMerge	该示例演示了应该如何通过自己的合并逻辑来创造一个电视指南显示。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 2 CustomMerge2	该示例演示了应该如何通过自己的合并逻辑来将合并模式混合到一起。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 3 CustomMerge3	该示例演示了应该如何通过重写“获取数据”GetData 方法来自定义分组。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义合并 4 CustomMerge4	该示例演示了应该如何通过重写“获取数据”GetData 方法来自定义分组。该示例调用了 C1.Win.C1FlexGrid 命名空间
自定义打印 CustomPrint	该示例演示了应该如何使用“创建图像”createImage 方法来打印含有任意行和列的间隔的表格。该示例使用了 C1FlexGrid 控件。

自定义打印多个表格 CustomPrintMultiGrid	该示例演示了应该如何使用“创建图像”createImage 方法来在一个单一的文件中打印多个表格。
自定义选择 CustomSelection	该示例使用了“所有者绘制”OwnerDraw 来实现自定义（非矩形）的选择。该示例调用了 C1.Win.C1FlexGrid 命名空间。
自定义排序 CustomSort	该示例演示了应该如何通过一个自定义的 IComparer 对象来实现与自己的排序逻辑。该示例使用了 C1FlexGrid 控件。
数据索引 DataIndex	该示例演示了应该如何使用“行数据索引”Row.DataIndex 属性来获取该行的基础数据。该示例使用了 C1FlexGrid 控件。
数据表 DataTable	该示例演示了应该如何绑定到一个“数据表”DataTable，应该如何添加和删除行。该示例使用了 C1FlexGrid 控件。
数据树型图 DataTree	该示例演示了应该如何将表格绑定到分层数据源上，并且将细节显示在子表格上。该示例调用了 C1.Win.C1FlexGrid 命名空间。
数据库动态样式 DBDynamicStyles	该示例可以根据其内容来指定样式到表格的单元格。该示例使用了 C1FlexGrid 控件。
数据库图像区域 DBImageField	该示例演示了应该如何显示“数据表”DataTable 中存储的图像。该示例使用了 C1FlexGrid 控件
数据库图像 DBImages	该示例演示了应该如何将一个表格通过存储为 OLE 对象的图像区域来绑定到一个数据库。该示例使用了 C1FlexGrid 控件。
数据库架构更改 DBSchemaChange	该示例演示了应该如何测试表格对数据源对象发生变化的响应。该示例使用了 C1FlexGrid 控件。
数据库树型图 DBTree	该示例说明了应该如何在一个分层树型视图中显示绑定的数据。该示例使用了 C1FlexGrid 控件。
数据库更新	该示例演示了应该如何将更改保存到底层

DBUpdate	数据库。该示例使用了 C1FlexGrid 控件。
拖放 DragDrop	该示例演示了应该如何自定义自动拖放。该示例使用了 C1FlexGrid 控件。
拖动图像 DragImages	该示例演示了应该如何通过自定义 OLE 拖放来使用单元格图像。
拖动合并 DragMerged	该示例演示了应该如何拖动合并列的分组。该示例使用了 C1FlexGrid 控件。
拖动行 DragRow	该示例演示了应该如何在表格之间拖动行。该示例使用了 C1FlexGrid 控件。
动态样式 DynamicStyles	该示例演示了应该如何以另一个单元格的内容为基础来指定样式到一个单元格。该示例使用了 C1FlexGrid 控件。
错误信息 ErrorInfo	该示例演示了应该如何在表格上显示错误信息。
Excel	该示例演示了应该如何加载和保存 Microsoft Excel 文件（.XLS）。
Excel 样式过滤器 ExcelStyleFilter	该示例添加了一个上下文菜单，它可以允许排序和过滤数据。
过滤行 FilterRow	该示例演示了应该如何在一个 FlexGrid 控件上实现一个“过滤行” FilterRow。该示例使用了 C1FlexGrid 控件。
查找行 FindRow	该示例演示了应该如何如何查找到一个在底层数据源的行。该示例使用了 C1FlexGrid 控件。
FlexByRow	该示例演示了应该如何建立一个“换位”格式的表格。该示例使用了 C1FlexGrid 控件。
FlexGroup	该示例演示了应该如何通过使用 C1FlexGrid 控件来实现 Outlook 风格的分组。该示例使用了 C1FlexGrid 控件。
固定底部 FreezeBottom	该示例演示了应该如何在表格的底部显示固定的行。该示例使用了 C1FlexGrid 控件。
GridChart	该示例演示了应该如何将一个 FlexGrid 附

	加到一个 C1Chart 控件。该示例使用了 C1FlexGrid 控件和 C1Chart 控件。
HierData	该示例演示了应该如何绑定到分层数据源（主从风格）。该示例使用了 C1FlexGrid 控件。
HierData2	该示例演示了应该如何绑定到分层数据源（主从风格）。
宿主控件 HostControls	该示例演示了应该如何在表格单元格中承载控件。该示例使用了 C1FlexGrid 控件。
超链接 Hyperlink	该示例演示了应该如何将超链接添加到 FlexGrid 中的单元格。该示例使用了 C1FlexGrid 控件。
遗产日期 LegacyDates	该示例演示了应该如何使用一个绑定的列来将存储为字符串的日期转换成真正的日期。
锁定的列 LockedColumns	该示例显示了应该如何防止用户选择某些列。
主从 MasterDetail	该示例演示了应该如何在代码中创建和绑定主从风格的数据集 DataSet。该示例调用了 C1.Win.C1FlexGrid 命名空间。
合并样式 MergeStyles	该示例演示了应该如何将分配到行、列和单元格的“单元格样式” CellStyles 合并。该示例使用了 C1FlexGrid 控件。
多列词典 MultiColumnDictionary	该示例演示了应该如何使用“多列词典” MultiColumnDictionary 类来实现多列下拉菜单。
多个家长 MultiParent	该示例演示了应该如何创建一个复杂的主从显示。
多个选择 MultiSelection	该示例演示了应该如何实现多范围的选择。
新功能 20091 NewFeatures20091	该示例显示了为 V1/2009 版本添加的新功能。
新的行模板	该示例演示了应该如何将一个新行模板放

NewRowTemplate	置到表格上。
大纲 Outline	该示例演示了应该如何隐藏大纲中的重复数据。该示例使用了 C1FlexGrid 控件。
自绘 OwnerDraw	该示例演示了应该如何自动调整自绘单元格的大小。
自绘阿尔法 OwnerDrawAlpha	该示例演示了应该如何使用有幻灯演示 transparency 的“自绘” OwnerDraw 来取得 MediaPlayer 一般的效果。
密码字符 PasswordChar	该示例显示了应该如何使用 C1FlexGrid 来输入和编辑密码。该示例使用了 C1FlexGrid 控件。
Pdf 输出 PdfExport	该示例演示了应该如何使用 C1Pdf 方法来将 C1FlexGrids 导出为 PDF 文件。
行状态显示 RowStateDisplay	该示例演示了应该如何使用不同的风格来显示不同状态的“数据行” DataRow。该示例使用了 C1FlexGrid 控件。
RTF 表格 RTFGrid	该示例演示了应如何显示表格单元格中的 RTF 文本。该示例使用了 C1FlexGrid 控件。
滚动位置 ScrollPosition	该示例演示了 ScrollPosition 属性是如何工作的。该示例使用了 C1FlexGrid 控件。
选择模式 SelectionMode	该示例演示了应该如何显示“选择模式” SelectionMode 属性的效果。该示例使用了 C1FlexGrid 控件。
设置编辑器 SetupEditor	该示例演示了应该如何使用“设置编辑器” SetupEditor 事件来自定义表格编辑器。该示例使用了 C1FlexGrid 控件。
简单计算器 SimpleCalc	该示例可以使用“数据表计算” DataTable.Compute 方法来评估表格单元格中的表达，并使用“自绘” OwnerDraw 来显示结果。
排序 Sorting	该示例演示了应该如何显示一些排序技术。该示例使用了 C1FlexGrid 控件。

排序空值 SortNulls	该示例演示了应该如何使用自定义排序来排列在表格底部的空值。
拆分 Splits	该示例演示了应该如何将一个 C1FlexGrid 拆分成多个视图。该示例使用了 C1FlexGrid 控件。
SQL 生成器 SqlBuilder	该示例演示了应该如何使用一个以表格为基础的图形界面来创建 SQL 语句。
开始编辑 StartEditing	该示例演示了应该如何实现一个停留在单元格编辑模式的表格。该示例使用了 C1FlexGrid 控件。
分类汇总 Subtotals	该示例演示了应该如何创建多个列的分类汇总。该示例使用了 C1FlexGrid 控件。
主题 Themes	该示例演示了应该如何通过更改表格的外观来实现几个预定义的展示。
线性更新 ThreadedUpdate	该示例显示了应该如何从不同的线程来更新表格。
树型图检查 TreeCheck	该示例演示了应该如何添加复选框到一个表格树型图。该示例使用了 C1FlexGrid 控件。
树型图节点 TreeNode	该示例演示了应该如何使用 FlexGrid 节点对象来管理大纲树型图。该示例使用了 C1FlexGrid 控件。
三态绑定 TriStateBound	该示例演示了应该如何在布尔列中提供三态（grayable）的值。该示例调用了 C1.Win.C1FlexGrid 命名空间。
未绑定的列 UnboundColumns	该示例演示了应该如何额外支持未绑定的列。
变焦单元格 ZoomCell	该示例演示了应该如何放大选定的单元格。

- ComponentOne FlexGrid for Mobile Devices Visual Basic 示例

示例	描述
分析	该示例演示了应该如何提供动态数据排序

Analyze	和分组。该示例使用了 C1FlexGrid 控件。
条形图表 BarChart	该示例演示了应该如何使用表格单元格来绘制条形图表。该示例使用了 C1FlexGrid 控件。
编辑数据 EditData	该示例演示了应该如何显示内置编辑器的不同样式和动态格式。该示例使用了 C1FlexGrid 控件。
线性更新 ThreadedUpdate	该示例显示了应该如何从不同的线程来更新表格。

- ComponentOne FlexGrid for Mobile Devices C# 示例

示例	描述
分析 Analyze	该示例演示了应该如何提供动态数据排序和分组。该示例使用了 C1FlexGrid 控件。
条形图表 BarChart	该示例演示了应该如何使用表格单元格来绘制条形图表。该示例使用了 C1FlexGrid 控件。
上下文菜单 ContextMenu	该示例演示了应该如何在选定表格上的一个项目后显示一个“上下文菜单”ContextMenu。
数据树型图 DataTree	该示例演示了应该如何在一个向下钻取的表格上显示分层数据。
编辑数据 EditData	该示例演示了应该如何显示内置编辑器的不同样式和动态格式。
主从 MasterDetail	该示例演示了应该如何使用两个表格来显示等级的数据。
透明图像 TransparentImages	该示例演示了应该如何使用透明度来诠释图像。

6. FlexGrid for WinForms 教程

下面章节的教程中，包含了一些例子用来说明 C1FlexGrid 控件中的一些主要特点。教程通过一步步创建几个简单的项目，详细描述了每个步骤。配套的光盘 CD 中包含了一些可作为参考的更复杂的例子。

这些教程是：

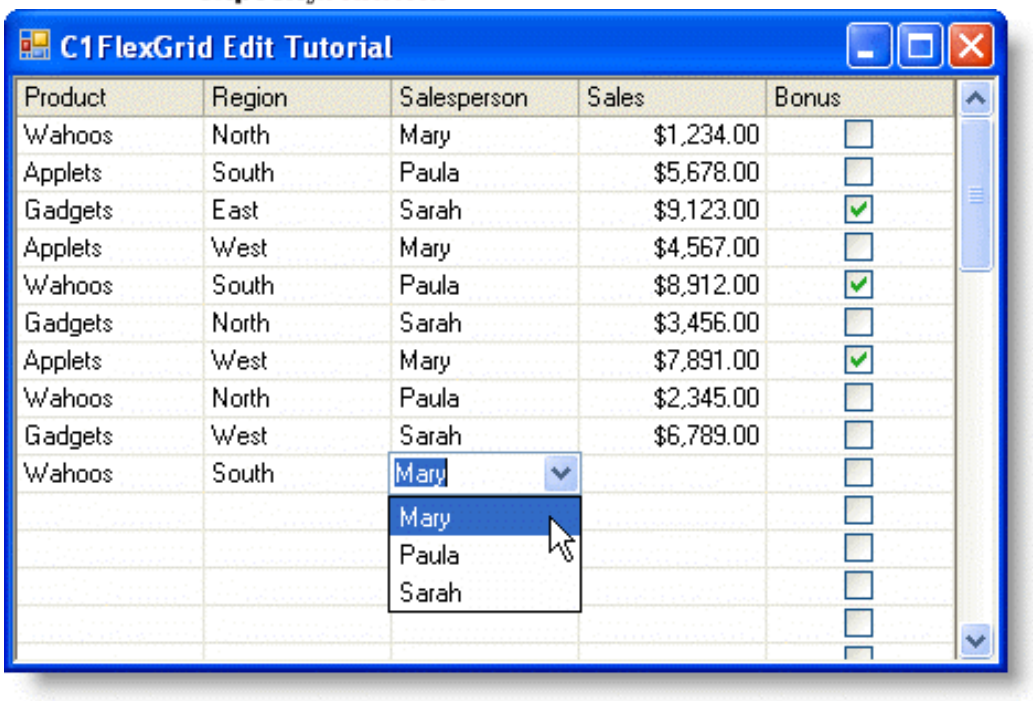
教程	描述
编辑教程	从一个基本的数据输入表格开始， 本教程演示如何实现数据格式化、复选框、下拉列表、输入掩码、数据验证、剪贴板支持。
大纲教程	演示如何使用 C1FlexGrid 的大纲功能来展示结构性（或分层）数据。
数据分析教程	从一个包含不同产品的、各地区的、销售人员的销售数据表格，来展示如何实现动态布局（列顺序）、自动排序、合并单元格、自动分类汇总和大纲视图。

6.1 编辑教程

本教程由一个基本的数据录入表格开始，然后增加了以下功能：

1. 格式化
2. 复选框
3. 下拉列表
4. 复杂的数据验证
5. 剪贴板支持
6. 自定义编辑器

最终的应用程序将类似于下面：



在 ComponentOne 主页的 Videos 中有一个本教程的视频。

6.1.1 步骤 1/6： 为这个编辑教程创建一个 C1FlexGrid 控件

启动一个新项目并点击工具箱中的 **C1FlexGrid** 图标向窗体中添加 C1FlexGrid 控件。

然后在窗体中点击或者拖动控件到一个合适的大小。

如果你没有在工具箱中找到 C1FlexGrid 控件，那么请在工具箱中右键单击然后选择“**选择项**...”。 然后在.Net 组件列表中查找 C1FlexGrid 控件并确保勾选上。 如果你在控件列表中无法找到控件，你也许需要重新安装控件包。

1. 在属性窗口中为 C1FlexGrid 控件设置以下属性：

属性	设置
Dock	Fill
Cols.Count	5
Cols.Fixed	0

2. 双击窗体的标题区域来打开代码窗口。在文件的顶端添加如下声明代码：

- Visual Basic

```
Imports C1.Win.C1FlexGrid
```

- C#

```
using C1.Win.C1FlexGrid;
```

这使得 C1FlexGrid 中定义的对象在整个项目都可见，并节省了很多需要敲的代码。

3. 可以通过列任务菜单、C1FlexGrid 列编辑器或者代码中来设计这些列。

在设计器中：

- 选中表格中的第一列。这将会打开这一列的**列任务**菜单。
- 在列标题和数据字段框中，输入 **Product**。
- 设置列标题和数据字段，其余列如下：

Column 1	
Column Caption	Region
Data Field	Region
Column 2	
Column Caption	Salesperson
Data Field	Salesperson
Column 3	
Column Caption	Sales
Data Field	Sales
Column 4	
Column Caption	Bonus
Data Field	Bonus

另外，还可以通过 **C1FlexGrid 列编辑器**来设置这些列：

- 通过在 C1FlexGrid 列任务菜单中点击设计器打开 C1FlexGrid 列编辑器。关于如何访问 C1FlexGrid 列编辑器的更多细节请查看访问 C1FlexGrid 列编辑器章节 （第 143 页）。
- 在右侧的窗格中选中 Column 0 。
- 在左侧窗格中，设置 Name 和 Caption 属性为 Product。
- 设置 Name 和 Caption 属性，其余列如下：

Column 1	
Name	Region
Caption	Region
Column 2	

Name	Salesperson
Caption	Salesperson
Column 3	
Name	Sales
Caption	Sales
Column 4	
Name	Bonus
Caption	Bonus

完成后点击 **OK** 来关闭编辑器。

在代码中：

在 **Form_Load** 事件中添加如下代码：

- Visual Basic

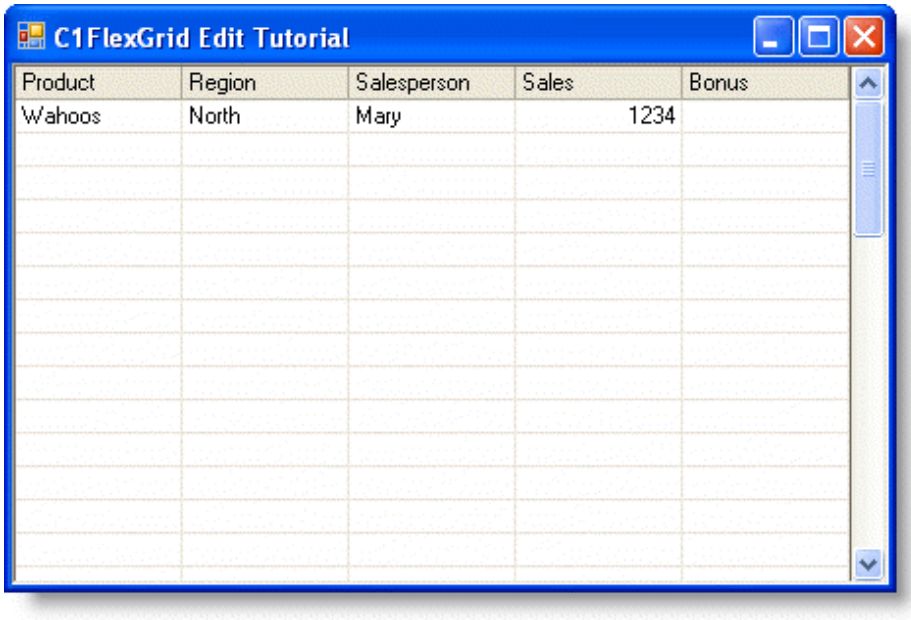
```
' 创建列.
Dim cols As String = "Product|Region|Salesperson|Sales|Bonus"
Dim colNames As String() = cols.Split("|")
Dim i%
For i = 0 To C1FlexGrid1.Cols.Count - 1
C1FlexGrid1(0, i) = colNames(i)
C1FlexGrid1.Cols(i).Name = colNames(i)
Next
```

- C#

```
//创建列.
string cols = "Product|Region|Salesperson|Sales|Bonus";
string[] colNames = cols.Split(new char[] { '|' });
for (int i = 0; i <= this.c1FlexGrid1.Cols.Count - 1; i++)
{
c1FlexGrid1[0, i] = colNames[i];
c1FlexGrid1.Cols[i].Name = colNames[i];
}
```

运行程序并遵守以下规定：

就是这样！按下 F5 运行该项目，这样你就可以开始向控件中输入一些数据。按 F2 或者空格键来编辑现有条目，或者只是在现有的条目上键入新数据。



6.1.2 步骤 2/6： 设置列的类型和格式

当显示数字或者日期值时，您通常会希望采用一致的格式来显示这些值。C1FlexGrid 控件允许你为每一列定制数据类型和格式。这些属性在设计器或者代码中都可以设置。

在设计器中

1. 在表格中选中 Sales 列。这将会打开 Sales 列的**列任务**菜单。
2. 设置 Data Type （数据类型）为 Decimal。
3. 在 **Format String**（格式字符串）框中点击**省略号**按钮打开对话框。
4. 在**格式类型**中选择 **Currency**（货币）。
5. 点击 **OK** 关闭格式字符串对话框。
6. 在表格中选中 **Bonus** 列。这将会打开 Bonus 列的**列任务**菜单。
7. 设置数据类型为 Boolean。

另外，还可以通过 **C1FlexGrid 的列编辑器**来设置数据类型和格式属性：

1. 通过 **C1FlexGrid 任务菜单**选择**设计器**来打开 **C1FlexGrid 列编辑器** 。 对于如何访问 **C1FlexGrid 列编辑器**的更多细节，请查看 访问 C1FlexGrid 列编辑器（第 143 页）章节。
2. 在右边的窗格中选择 Sales 列。
3. 设置 DataType 属性为 Decimal。
4. 点击 **Format** 属性旁边的**省略号**按钮。这将会打开 **Format String** （格式

化字符串）对话框。

5. 在**格式化**类型 中选择**货币**。
6. 点击 **OK** 来关闭 **Format String** 对话框。
7. 在右边的窗格中选择 Bonus 列。
8. 设置 DataType 属性为 Boolean。
9. 点击 **OK** 来关闭编辑器窗口。

代码中

要指定列的数据类型和格式，先添加步骤 1/6：为这个编辑教程创建一个 C1FlexGrid 控件（第 108 页）的代码后再添加下面的代码：

- Visual Basic

```
'设置列的数据类型和格式。
Dim c As Column = C1FlexGrid1.Cols("Sales")
c.DataType = GetType(Decimal)
'  货币类型.
c.Format = "c2"
c = C1FlexGrid1.Cols("Bonus")
c.DataType = GetType(Boolean)
c.ImageAlign = ImageAlignEnum.CenterCenter
```

- C#

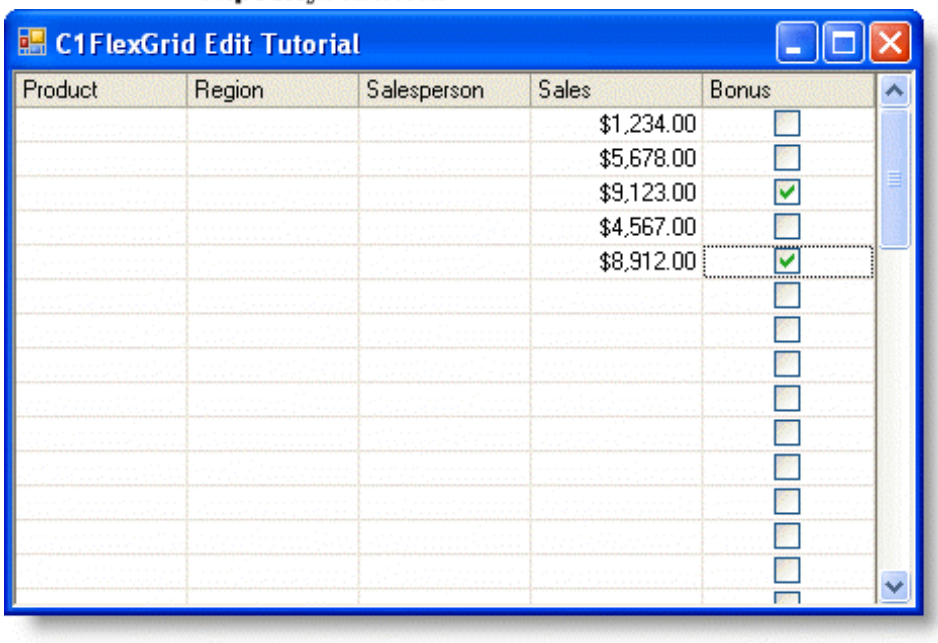
```
//设置列的数据类型和格式。
Column c = c1FlexGrid1.Cols["Sales"];
c.DataType = typeof(Decimal);
// 货币类型.
c.Format = "c2";
c = c1FlexGrid1.Cols["Bonus"];
c.DataType = typeof(bool);
c.ImageAlign = ImageAlignEnum.CenterCenter;
```

运行程序，并遵守以下规定：

Sales 列新的代码格式，用来存储和显示货币值， Bonus 列用来处理布尔值。

如果你在 Sales 列输入一些数字和非数字值，你会发现表格将无法接受这些输入。

Bonus 列显示成复选框的样子，可以用鼠标、键盘来切换，这是布尔值列的默认行为。



注意格式化属性，它不以任何方式影响数据本身的值，只影响如何显示。

6.1.3 步骤 3/6：纳入下拉列表

录入数据是一个乏味和容易出错的过程。下拉列表是个伟大的发明，因为它极大的减少了你必须做的输入操作，同时也就减少了出错的机会，这样就提高了数据的一致性。

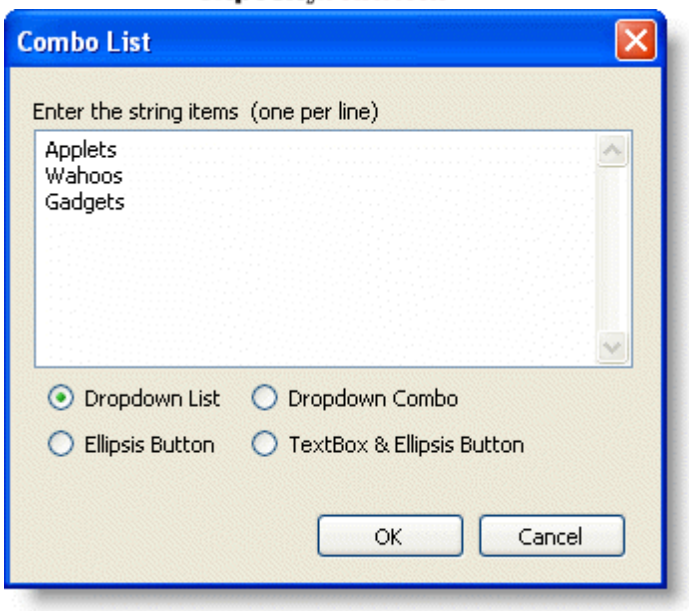
让我们假设我们的教程项目只涉及三种产品的销售（**Applets、Wahoos、和 Gadgets**）。在四个区域（东、南、西、北），并且这有三名全职销售人员（玛丽、莎拉和宝拉）。

要在 C1FlexGrid 控件中将这个列表变为下拉列表，所有你必须做的就是分别建立包含了每个项目的字符串（使用字符分割），并将它们分配到每个列对应的 ComboList 属性中去。

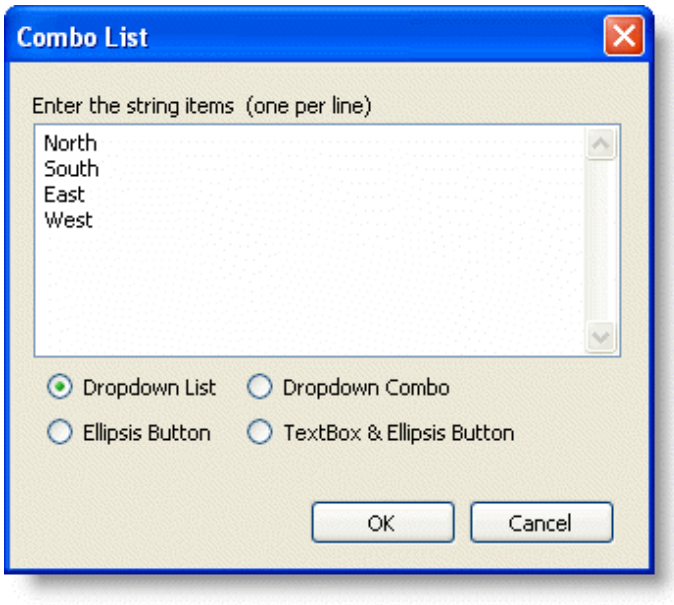
这个属性可以在代码或者设计器中设置。

在设计器中：

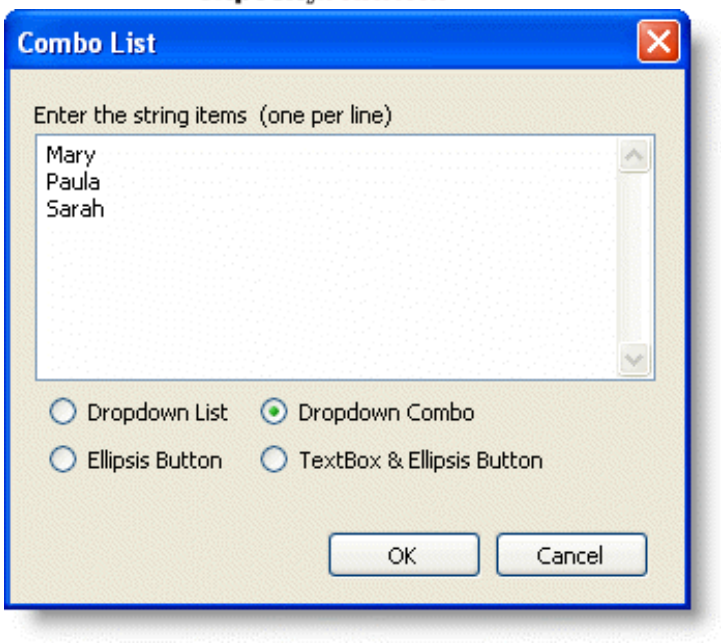
1. 选中 Product 列。 这将会打开 Product **列事件**菜单对话框。
2. 在 Combo List 框中点击省略号按钮来打开 Combo List 窗口。
3. 如下所示，输入 Applets、Wahoos、和 Gadgets。



4. 点击 OK 来关闭 **Combo List** 对话框。
5. 选中 Region 列。
6. 在 Combo List 框中点击省略号 按钮。
7. 输入 North、South、East、和 West 如下图所示：



8. 点击 **OK**。
9. 选中 Salesperson 列。
10. 在 **Combo List** 框中点击**省略号** 按钮。
11. 输入 Mary、Paula、和 Sarah 如下图所示：



12.选择 Dropdown Combo（下拉组合框）选项。

13.点击 OK。

另外，ComboList 属性同样可以在 **C1FlexGrid 列编辑器**中设置：

1. 通过在 **C1FlexGrid** 任务菜单中选择设计器打开 **C1FlexGrid 列编辑器**。想要知道对于如何访问 **C1FlexGrid 列编辑器**的更多细节，请查看 访问 C1FlexGrid 列编辑器（第 143 页）章节。

2. 在右边的窗格中选择 Product 列 。

3. 在左边的窗格中，设置 ComboList 属性为 Applets|Wahoos|Gadgets。

4. 在右边的窗格中选择 Region column。

5. 在左边的窗格中，设置 ComboList 属性为 North|South|East|West。

6. 在右边的窗格中选择 Salesperson。

7. 在左边的窗格中，设置 ComboList 属性为|Mary|Paula|Sarah。

8. 点击 OK 来关闭编辑器.

在代码中：

要添加下拉列表项目，先添加步骤 2/6：设置列的类型和格式（第 111 页）的代码后再添加下面的代码：

- Visual Basic

```
'设置下拉列表
C1FlexGrid1.Cols("Product").ComboList = "Applets|Wahoos|Gadgets"
C1FlexGrid1.Cols("Region").ComboList = "North|South|East|West"
C1FlexGrid1.Cols("Salesperson").ComboList = "|Mary|Paula|Sarah"
```

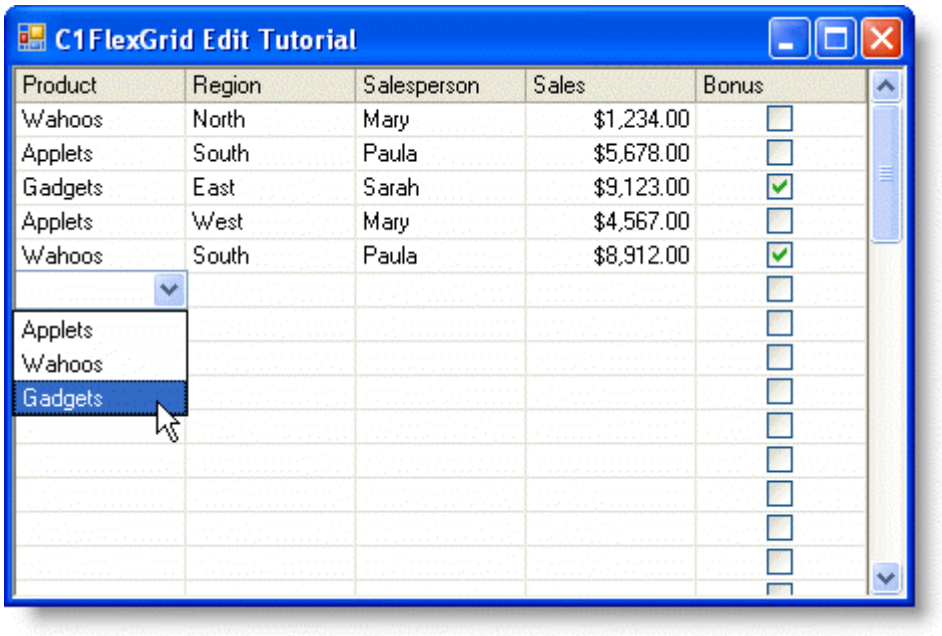
- C#

```
'设置下拉列表
c1FlexGrid1.Cols["Product"].ComboList = "Applets|Wahoos|Gadgets";
c1FlexGrid1.Cols["Region"].ComboList = "North|South|East|West";
c1FlexGrid1.Cols["Salesperson"].ComboList = "|Mary|Paula|Sarah";
```

运行程序，并遵守以下规定：

请注意最后一个下拉列表是一个入力框（文本入力光标）。这允许用户输入不在列表中的其他名字。换句话说，这些值将在下拉列表中可以编辑，而不是一个简单的下拉列表。

按 F5 再次运行该项目，然后移动光标。当您将光标移动到有下拉列表的列上时，下拉列表的按钮才为可见，你可以点击它来显示列表，或者只需要简单输入某个条目的第一个字母，使这个条目在下拉列表中变为高亮。



6.1.4 步骤 4/6： 添加数据验证

如果你为表格中的列指定了数据类型，表格就会确保只有这个类型的数据能够在这一列保存。这有助于防止错误发生，但是你会经常需要更加严格的验证，以确保输入的数据是正确的。对于这一点，你应该使用 ValidateEdit 事件。

例如，想象一下，反托拉斯法规组织我们在北方地区销售我们的小程序。为了防止数据输入错误而带来昂贵的诉讼，我们希望防止用户在这个下拉列表中输入这个值。

以下事件用来检查在数据输入后编辑框中的数据是否合法， 添加下面的代码到窗体中：

- Visual Basic

```
Private Sub C1FlexGrid1_ValidateEdit(ByVal sender As Object, ByVal e As
ValidateEditEventArgs) Handles C1FlexGrid1.ValidateEdit
Dim rgn As String = String.Empty
Dim prd As String = String.Empty
' 收集我们需要的数据
Select Case e.Col
prd = C1FlexGrid1.Editor.Text
rgn = C1FlexGrid1(e.Row,          "Region")
Case        1
prd = C1FlexGrid1(e.Row,          "Product")
rgn = C1FlexGrid1.Editor.Text
End Select
' 我们可以在北部区域卖小程序
If prd = "Applets" And rgn = "North" Then
MsgBox("Warning: Regulation #12333AS/SDA-23 forbids " & _
"the sale of " & prd & " in region " & rgn & ". " & _
"Please verify input.")
e.Cancel = True
End If
End Sub
```

- C#

```
        private void c1FlexGrid1_ValidateEdit( object sender,
ValidateEditEventArgs e)
        {
        string rgn = string.Empty;
        string prd = string.Empty;
        //收集我们需要的数据.
        switch (e.Col)
        {
        case 0:
        prd = c1FlexGrid1.Editor.Text;
        rgn = (string)c1FlexGrid1[e.Row, "Region"];
        break;
        case 1:
        prd = (string)c1FlexGrid1[e.Row, "Product"];
        rgn = c1FlexGrid1.Editor.Text;
        break;
        }
        //我们可以在北部区域卖小程序
        if ( prd == "Applets" && rgn == "North" ) {
        MessageBox.Show("Warning: Regulation #12333AS/SDA-23 forbids " +
        "the sale of " + prd + " in region " + rgn + ". " +
        "Please verify input.");
        e.Cancel = true;
        }
        }
```

运行程序，并遵守以下规定：

这个模块开始是收集需要验证的输入。请注意，被检查的值要恢复的话请使用 Editor.Text 属性。 这是必要的，因为这次编辑并没有应用到控件中去。如果检查失败了，这个模块将会显示一个警告然后将 Cancel 参数设置为 **True**，这样可以取消编辑并将单元格重新置为编辑模式，使用户可以再次尝试。

按 F5 再次运行该项目，然后再尝试输入一些错误的值。你会看到该控件将会拒绝它们。

6.1.5 步骤 5/6： 添加剪贴板的支持

Windows 剪贴板是一个在应用程序之间传输信息的非常有用的设备。

在 **FlexGrid for WinForms** 项目中添加剪贴板支持是相当容易的。只需要在代码或者设计器中简单的设置 AutoClipboard 属性为 **True** ，然后表格就会自动的处理所有发送到剪贴板的标准键盘命令：如 CTRL+X 或者 SHIFT+DELETE 剪切，CTRL+C 或者 CTRL+INSERT 拷贝，CTRL+V 或者 SHIFT+INSERT 粘贴。

在设计器中：

在属性窗口中定位到 **AutoClipboard** 属性并将它设置为 **True**。

在代码中：

在添加完步骤 3/6：纳入下拉列表（第 112 页）之后，添加如下代码：

- Visual Basic

```
C1FlexGrid1.AutoClipboard = True
```

- C#

```
c1FlexGrid1.AutoClipboard = true;
```

另一个伟大的 Windows 功能，并与剪贴板密切相关，这就是 OLE 拖放操作。C1FlexGrid 有两个属性， DragMode 和 DropMode。 这两个属性实现了这个功能。无论是在代码中还是在设计器中，只需要将这个两个属性设置为自动，你就能够将表格中选中的项目拖出程序并放置到其他应用程序中，如 Microsoft Excel，或者从 Excel 中选择一个范围拖放到 C1FlexGrid 控件中。

在设计器中：

定位到 DragMode 和 DropMode 属性并将它们都设置为 Automatic。

在代码中：

在设置 **AutoClipboard** 属性后添加如下代码：

- Visual Basic

```
C1FlexGrid1.DragMode = DragModeEnum.Automatic
C1FlexGrid1.DropMode = DropModeEnum.Automatic
```

- C#

```
c1FlexGrid1.DragMode = DragModeEnum.Automatic;
c1FlexGrid1.DropMode = DropModeEnum.Automatic;
```

运行程序，并遵守以下规定：

按 F5 再次运行该项目，然后试着拷贝和粘贴一些数据。注意，你可以粘贴一些非法的数据，因为我们的粘贴操作不触发任何的数据验证事件。 这是留给读者的练习。

6.1.6 步骤 6/6： 包含一个自定义编辑器

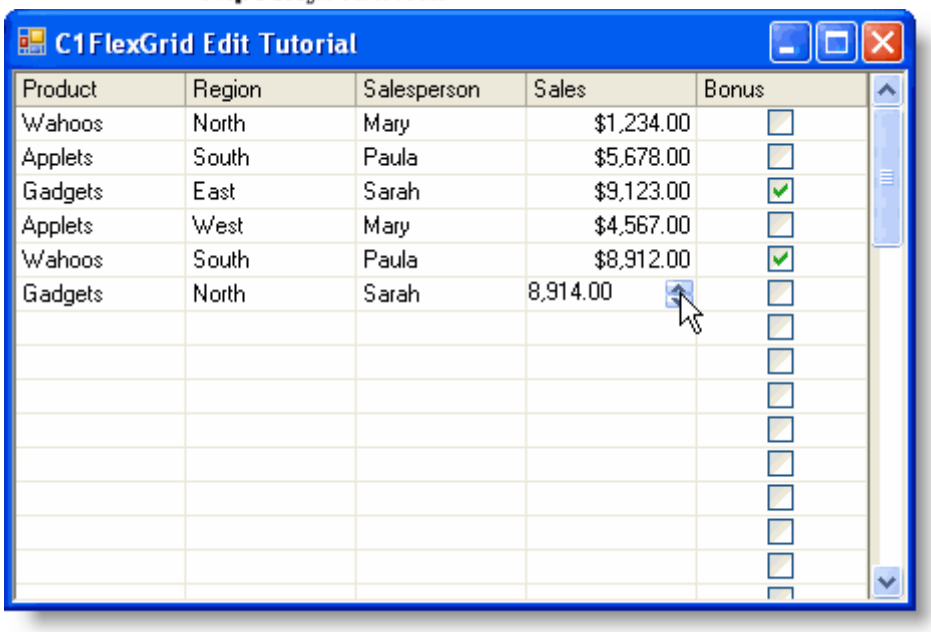
C1FlexGrid 拥有强大的内置输入编辑器：输入文字、文字掩码、列表选择、复选框等等。

但是在某些情况下，你可能想使用一个自定义编辑器来代替，也许是 **C1Input** 库中带的输入控件或者你自己写的输入控件。从版本 2.5 开始，FlexGrid 允许你轻松加愉快的插入自定义编辑器。要在教程工程的 Sales 列附加一个自定义编辑器，从往窗体上添加一个 **NumericUpDown** 控件开始。

在属性窗口中，为 **NumericUpDown** 控件设置如下属性：

属性	设置
BorderStyle	None
DecimalPlaces	2
Maximum	5000000
ThousandsSeparator	True
Visible	False

在 C1FlexGrid 任务菜单中选择设计器来打开表格的列编辑器。



选中 Sales 列（或代表 Sales 列的第 3 列），之后设置编辑器属性为 **NumericUpDown**。

运行程序，并遵守以下规定：

现在运行该项目，并尝试在 Sales 列中编辑一些值。请注意，表格现在使用 **NumericUpDown** 控件，而不是内置的编辑器。

这个控件已经进行了适当的位置调整、初始化和管理。当你将光标的焦点移动到不同的单元格中时，这个值将会保存到表格中。

但是这个自定义编辑器的行为与内置编辑器并不完全一样。例如：

1. 当你开始编辑并键入一个数字，旧的值不会被清除。
2. 编辑器的大小是不完全正确的（他看起来有点太小）。
3. 当你使用回车键完成编辑时会有一声提示音出现。

你可以使用几个方法来克服这些问题。一种方法是使用编辑器的事件，但是这将使其难以在其他项目中重用这个控件。另一个方法是创建一个派生类并实现 I**C1EmbeddedEditor** 接口的一些方法，类似下面的这些代码：

- Visual Basic

```
Public Class MyUpDown
Inherits NumericUpDown
' 设置初始值
Public Sub C1EditorInitialize(ByVal value As Object, ByVal
editorAttributes As IDictionary)
' 为初始值赋值
value = Convert.ChangeType(value, GetType(Decimal))
' 选择进入点
MyBase.Select(0, Int32.MaxValue)
End Sub
' 设置字体高度以便控件来控制整体高度
Public Sub C1EditorUpdateBounds(ByVal rc As Rectangle)
MyBase.FontHeight = rc.Height
Bounds = rc
End Sub
' 当用户按下回车键，禁止发出蜂鸣声
Protected Overrides Function ProcessDialogKey(ByVal keyData As Keys) As
Boolean
If (keyData = Keys.Enter) Then
Parent.Focus()
If (Parent.Focused) Then SendKeys.Send("{Down}")
Return True
End If
Return MyBase.ProcessDialogKey(keyData)
End Function
```

- C#

```
internal class MyUpDown : NumericUpDown
{
//设置初始值
public void C1EditorInitialize(object value,
System.Collections.IDictionary editorAttributes)
{
//为初始值赋值
Value = (decimal)Convert.ChangeType(value, typeof(decimal));
//选择进入点
Select(0, int.MaxValue);
}
//设置字体高度以便控件来控制整体高度
public void C1EditorUpdateBounds(Rectangle rc)
{
base.FontHeight = rc.Height;
Bounds = rc;
}
//当用户按下回车键，禁止发出蜂鸣声
override protected bool ProcessDialogKey(Keys keyData)
{
if (keyData == Keys.Enter)
{
Parent.Focus();
if (Parent.Focused) SendKeys.Send("{Down}");
return true;
}
return base.ProcessDialogKey(keyData);
}
}
```

上面的代码实现了三个方法：

C1EditorInitialize 被调用来初始化编辑器。它设置了初始值然后选中整个条目。

这个将解决刚才提到的第一个问题。因为整个项目被选中，现在输入一个我们想要的字符将会取代目前的内容。

C1EditorUpdateBounds 被调用来定位正在编辑的单元格。**NumericUpDown** 控件的高度被设为以字体大小为基础自动增长（这就是为什么它相对单元格看起来太短）。代码设置了编辑器的 **FontHeight** 属性，所以它将被固定在单元格中。

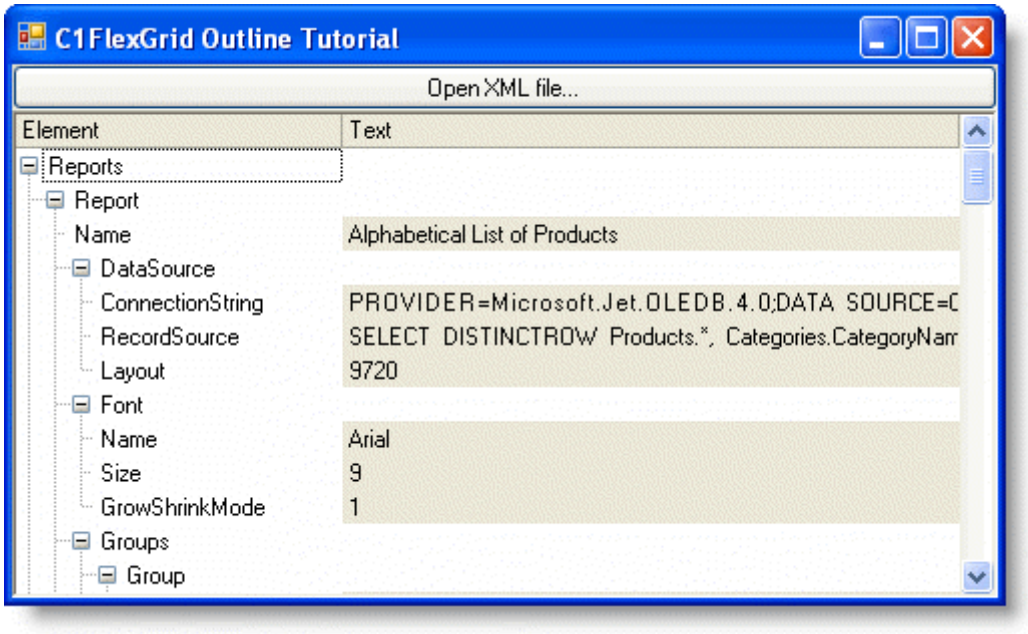
ProcessDialogKey 方法被重写了，用来阻止当用户按下回车键发出的提示音。

6.2 大纲教程

本教程演示如何使用 C1FlexGrid 作为一个大纲视图来显示结构（或层次）数据。

当作为一个大纲，C1FlexGrid 控件的行为就像一个树形控件，显示一些可以折叠、展开的节点，以展示包含下级数据的分支。

本教程可以让你加载一个 XML 文档到表格中，它将显示为一颗树。最终的应用程序看起来的效果如下：



6.2.1 步骤 1/5： 创建控件

新建一个项目并添加两个控件：

- 在窗体的顶部附近添加一个命令按钮。

- 在按钮下面的区域添加一个 C1FlexGrid 控件。

如果你没有在工具箱中找到 C1FlexGrid 控件，那么请在工具箱中右键单击然后选择“**选择项**...”。 然后在.Net 组件列表中查找 C1FlexGrid 控件并确保勾选上。 如果你在控件列表中无法找到控件，你也许需要重新安装控件包。

1. 在属性窗口中设置如下属性：

命令按钮

属性	设置
Dock	Top
Text	"Open XML File..."

C1FlexGrid

属性	设置
Dock	Fill

2. 双击窗体的标题区域来打开代码窗口。在文件的最上方添加如下代码段：

- Visual Basic

```
Imports C1.Win.C1FlexGrid
```

- C#

```
using C1.Win.C1FlexGrid;
```

这使得 C1FlexGrid 中定义的对象在整个项目都可见，并节省了很多需要敲的代码。

3. 可以在设计器中使用属性窗口和编辑器来设置表格，或者在代码中输入（或复制）以下代码：

在设计器中：

在属性窗口中为 C1FlexGrid 控件设置以下属性：

属性	设置
Cols.Count	2
Cols.Fixed	0
ExtendLastCol	True
Rows.Count	1
Tree.Column	0
Tree.Style	SimpleLeaf

为表格设置样式：

- 在 C1FlexGrid 任务菜单中选择样式打开 **C1FlexGrid 样式编辑器**。想要知道对于如何访问 **C1FlexGrid 样式编辑器**的更多细节，请查看 访问 C1FlexGrid 样式编辑器（第 143 页）章节。
- 在内置样式中选择 **Normal** 。
- 设置 Border.Style 为 None，设置 TextAlign 属性为 LeftCenter，设置 WordWrap 属性为 False。
- 点击**添加**按钮。
- 将 CustomStyle1 重命名为 Data。
- 设置 BackColor 属性为 Control。
- 点击 OK 来关闭编辑器。

设置表格中的列：

- 在表格中选中 Column 0。这将打开第 0 列的列任务菜单。
- 将 Column Caption 设置为 Element。
- 取消勾选 Allow Editing 复选框。
- 选择 Column 1.
- 将 Column Caption 设置为 Text。

另外，这些列也可以通过 **C1FlexGrid 列编辑器**来设置：

- 通过在 **C1FlexGrid 任务**菜单中选择**设计器**来打开 **C1FlexGrid 列编辑器**。想要知道对于如何访问 **C1FlexGrid 列编辑器**的细节，请查看访问 C1FlexGrid 列编辑器章节（第 143 页）。
- 在右侧窗格中选择 Column 0。
- 在左侧窗格中，将 AllowEditing 属性设置为 False ，设置 Caption 属性为 Element。
- 在右侧窗格中选择 **Column 1。**
- 在左侧窗格中，将 **Caption** 属性设置为 **Text。**
- 点击 OK 来关闭编辑器。

在代码中：

- Visual Basic

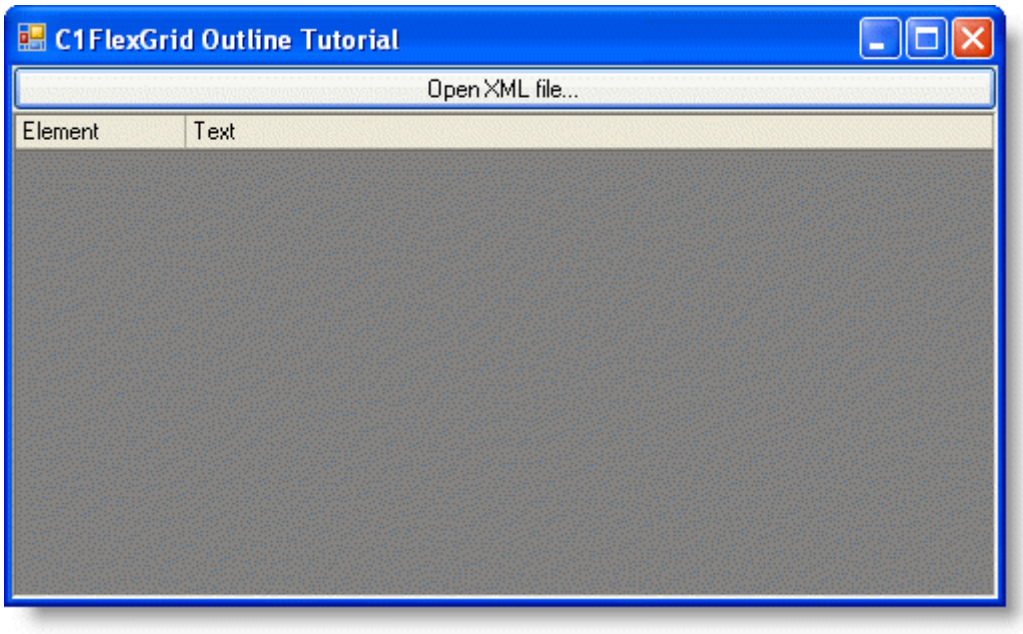
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 初始化表格
   C1FlexGrid1.Rows.Count = 1
C1FlexGrid1.Cols.Count = 2
C1FlexGrid1.Cols.Fixed = 0
C1FlexGrid1.ExtendLastCol = True
C1FlexGrid1(0, 0) = "Element"
C1FlexGrid1(0, 1) = "Text"
' 初始化大纲的树结构
C1FlexGrid1.Tree.Column = 0
C1FlexGrid1.Tree.Style = TreeStyleFlags.SimpleLeaf
C1FlexGrid1.Cols(0).AllowEditing = False
' 初始化样式
C1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.None
C1FlexGrid1.Styles.Normal.TextAlign = TextAlignEnum.LeftCenter
C1FlexGrid1.Styles.Normal.WordWrap = False
Dim cs As CellStyle = C1FlexGrid1.Styles.Add("Data")
cs.BackColor = SystemColors.Control
End Sub
```

- C#

```
private void Form1_Load( System.object sender, System.EventArgs e)
{
//初始化表格
c1FlexGrid1.Rows.Count = 1;
c1FlexGrid1.Cols.Count = 2;
c1FlexGrid1.Cols.Fixed = 0;
c1FlexGrid1.ExtendLastCol = true;
c1FlexGrid1[0, 0] = "Element";
c1FlexGrid1[0, 1] = "Text";
// 初始化大纲的树结构
c1FlexGrid1.Tree.Column = 0;
c1FlexGrid1.Tree.Style = TreeStyleFlags.SimpleLeaf;
c1FlexGrid1.Cols[0].AllowEditing = false;
//初始化样式
c1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.None;
c1FlexGrid1.Styles.Normal.TextAlign = TextAlignEnum.LeftCenter;
c1FlexGrid1.Styles.Normal.WordWrap = false;
CellStyle cs = c1FlexGrid1.Styles.Add("Data");
cs.BackColor = SystemColors.Control;
}
```

运行程序，并遵守以下规定：

代码从一开始就设置表格的布局和标题的文本内容。



接下来，初始化大纲树属性并将第一列的 AllowEditing 属性设为 False 以防止 XML 节点能够被编辑。请注意，用户仍可以在包含每个 XML 节点数据的下一列进行数据的编辑。

现在空间已经设置好了。我们可以开始添加一些代码。

6.2.2 步骤 2/5：读取数据并创建大纲

要读取数据，并建立大纲，向 Button1_Click 事件中添加一些代码和 GetXMLData 事物。

1. 双击命令按钮，将下面的代码添加到 Button1_Click 事件中去：

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
' 取得文件名称
Dim fo As OpenFileDialog = New OpenFileDialog()
fo.DefaultExt = "xml"
fo.Filter = "XML Files (*.xml)|*.xml"
If fo.ShowDialog() <> Windows.Forms.DialogResult.OK Then Exit Sub
' 读取 XML 文件
Dim xdoc As System.Xml.XmlDocument = New System.Xml.XmlDocument()
xdoc.Load(fo.FileName)
' 停止重绘，以提高速度。
C1FlexGrid1.Redraw = False
' 填充表格
C1FlexGrid1.Rows.Count = 1
GetXMLData(xdoc.ChildNodes(1), 0)
' 自动调整树的大小。
C1FlexGrid1.AutoSizeCol(0)
' 显示 0、1、2 层
C1FlexGrid1.Tree.Show(2)
' 开始重绘
C1FlexGrid1.Redraw = True
End Sub
```

- C#

```
private void button1_Click(System.object sender, System.EventArgs e)
{
//取得文件名称
OpenFileDialog fo = new OpenFileDialog();
fo.DefaultExt = "xml";
fo.Filter = "XML Files (*.xml)|*.xml";
if ( fo.ShowDialog() != DialogResult.OK ) return;
//读取 XML 文件
System.Xml.XmlDocument xdoc = new System.Xml.XmlDocument();
xdoc.Load(fo.FileName);
//停止重绘，以提高速度。
c1FlexGrid1.Redraw = false;
//填充表格
c1FlexGrid1.Rows.Count = 1;
GetXMLData(xdoc.ChildNodes[1], 0);
//自动调整树的大小。
c1FlexGrid1.AutoSizeCol(0);
//显示 0、 1、 2 层
c1FlexGrid1.Tree.Show(2);
//开始重绘
c1FlexGrid1.Redraw = true;
}
```

请遵守以下规定：

这个流程一开始是显示一个 **OpenFileDialog** 用来让用户选择一个 XML 文件加载到表格中。

当文件被选中，程序就开始加载文件到 **XmlDocument** 对象，并将文件中的内容解析到内存中去。

这个流程之后开始设置表格的重绘属性为 **False** 来控制表格暂停重绘。这种技显著的提高了程序的性能，你可以在向 C1FlexGrid 表格中添加大量数据的时候使用这个方法。

接下来，这个流程清除了为 1 的任何数据，并调用 **GetXMLData** 方法来填充数据到 XmlDocument。这个 **GetXMLData** 方法是这个指南中列出的条目中，最重要的一环。

在表格已被填充后，例程使用 AutoSizeCol 方法来根据第一列的内容来调整列的宽度，之后 Show 方法用来将需要显示的层级扩展到 0、 1、 和 2。 之后例

程将 Redraw 属性设置回 **True** 以便 grid 开始重绘。

2. 这个 GetXMLData 例程是本教程中最有趣的一个。它贯穿整个 XMLDocument 对象，并建立大纲的树形结构。将下面的代码添加到窗体中去：

- Visual Basic

```
Private Sub GetXMLData(ByVal node As System.Xml.XmlNode, ByVal level As
Integer)
' 跳过注释节点
If node.NodeType = System.Xml.XmlNodeType.Comment Then
Exit Sub
End If
' 为这一个节点添加一个新行
Dim row As Integer = C1FlexGrid1.Rows.Count
C1FlexGrid1.Rows.Add()
' 为这个新行添加数据
C1FlexGrid1(row, 0) = node.Name
If node.ChildNodes.Count = 1 Then
C1FlexGrid1(row, 1) = node.InnerText
C1FlexGrid1.SetCellStyle(row, 1, C1FlexGrid1.Styles("Data"))
End If
' 如果这个节点有"Name" 子节点的话，将它保存起来作为一个 ToolTip 提
示使用。
If node.ChildNodes.Count > 0 Then
Dim ndName As System.Xml.XmlNode = node.SelectSingleNode("Name")
If Not (ndName Is Nothing) Then
C1FlexGrid1.Rows(row).UserData = ndName.InnerText
End If
End If
' 如果此节点有子节点，将它们都取出来
If node.ChildNodes.Count > 1 Then
' 将当前行作为节点
C1FlexGrid1.Rows(row).IsNode = True
C1FlexGrid1.Rows(row).Node.Level = level
' 使用递归来取得子节点
Dim child As System.Xml.XmlNode
For Each child In node.ChildNodes
GetXMLData(child, level + 1)
Next
End If
End Sub
```

- C#

```
private void GetXMLData(System.Xml.XmlNode node, int level)
{
//跳过注释节点
if ( node.NodeType == System.Xml.XmlNodeType.Comment )
{
return;
}
//为这一个节点添加一个新行
int row = c1FlexGrid1.Rows.Count;
c1FlexGrid1.Rows.Add();
//为这个新行添加数据
c1FlexGrid1[row, 0] = node.Name;
if ( node.ChildNodes.Count == 1 )
{
c1FlexGrid1[row, 1] = node.InnerText;
c1FlexGrid1.SetCellStyle(row, 1, c1FlexGrid1.Styles["Data"]);
}
//如果这个节点有"Name" 子节点的话，将它保存起来作为一个 ToolTip 提示使用。
if (node.ChildNodes.Count > 0)
{
System.Xml.XmlNode ndName = node.SelectSingleNode("Name");
if (ndName != null)
{
c1FlexGrid1.Rows[row].UserData = ndName.InnerText;
}
}
//如果此节点有子节点，将它们都取出来
if ( node.ChildNodes.Count > 1 )
{
//将当前行作为节点
c1FlexGrid1.Rows[row].IsNode = true;
c1FlexGrid1.Rows[row].Node.Level = level;
//使用递归来取得子节点
foreach (System.Xml.XmlNode child in node.ChildNodes)
GetXMLData(child, level + 1);
}
}
```

请遵守以下规定：

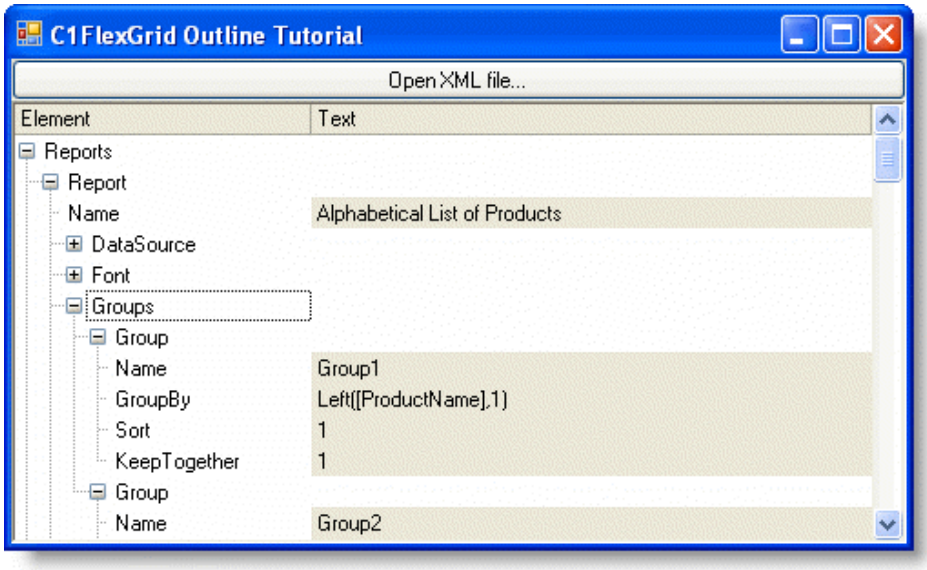
程序一开始跳过了 XML 注释节点，然后使用 **Rows.Add** 来向表格中添加一个新行。

接下来，程序开始设置节点的名称并检查该节点是否拥有孩子节点。在这种情况下，该节点被解释为一个数据节点，之后将该节点的 **InnerText** 属性复制到新行的第二列。代码还设置了当窗体被创建时，包含数据的单元格的样式为自定义的“Data”样式。接下来的代码块检查看看是否这个节点包含一个名为“name”的子节点。如果有，那么“name”节点的内容会被分配到新行的 UserData 属性中去。这个值将会被用于实施工具提示，这样即使节点名称被折叠起来了，用户也能够看到。

最后，如果该节点有孩子节点， **GetXMLData** 程序会调用自身来递归添加他们到表格中。

运行程序，并遵守以下规定：

该项目可以加载 XML 并显示它们，用户可以点击折叠或者展开节点。



接下来的步骤中会添加一些改进，使得应用程序更加容易使用。

6.2.3 步骤 3/5：添加自定义的鼠标和键盘处理

C1FlexGrid 为你提供了展开和折叠的操作，但是你可以扩展并自己定制这些行为。每当一个分支被展开或折叠起来，控件会触发 BeforeCollapse 和 AfterCollapse 事件，所以你可以在这些事件中做一些回应。此外，您可以在代码中使用 Collapsed 属性来获取和设置每个分支的折叠状态。

在本教程中，我们将捕获当用户鼠标双击或按下回车键来展开或者折叠节点时的 **DoubleClick** 和 **KeyPress** 事件。添加下面的 **DoubleClick** 和 **KeyPress** 事件代码到窗体上：

- Visual Basic

```
Private Sub C1FlexGrid1_DoubleClick(ByVal sender As Object, ByVal e As
EventArgs) Handles C1FlexGrid1.DoubleClick
If C1FlexGrid1.MouseCol = 0 Then
ToggleNodeState()
End If
End Sub
Private Sub C1FlexGrid1_KeyPress(ByVal sender As Object, ByVal e As
KeyPressEventArgs) Handles C1FlexGrid1.KeyPress
If e.KeyChar = vbCr Then
ToggleNodeState()
End If
End Sub
Private Sub ToggleNodeState()
' 如果当前行不是节点，则什么也不做
Dim r As Row = C1FlexGrid1.Rows(C1FlexGrid1.Row)
If Not r.IsNode Then Exit Sub
'切换折叠状态
r.Node.Collapsed = Not r.Node.Collapsed
End Sub
```

- C#

```
private void c1FlexGrid1_DoubleClick( object sender, EventArgs e)
{
if ( c1FlexGrid1.MouseCol == 0 )
{
ToggleNodeState();
}
}
private void c1FlexGrid1_KeyPress( object sender, KeyPressEventArgs e)
{
if ( e.KeyChar == 13 )
{
ToggleNodeState();
}
}
private void ToggleNodeState()
{
//如果当前行不是节点，则什么也不做
Row r = c1FlexGrid1.Rows[c1FlexGrid1.Row];
if (! r.IsNode ) return;
//切换折叠状态
r.Node.Collapsed = !r.Node.Collapsed;
}
```

运行程序，并遵守以下规定：

事件处理程序检查用户是否双击第一列或按下回车键，然后调用 **ToggleNodeState** 程序。**ToggleNodeState** 检查当前行是否是一个节点行，如果是，则切换 Collapsed 属性值。

6.2.4 步骤 4/5：允许/阻止编辑

回想一下，我们设置第一列的 AllowEditing 属性为 **False**。这可以防止用户编辑第一列中的任何单元格。我们也想防止用户在节点行的第二列输入任何数据。要做到这一点，添加下面的代码来处理 BeforeEdit 事件：

- Visual Basic

```
Private Sub C1FlexGrid1_BeforeEdit(ByVal sender As Object, ByVal e As
RowColEventArgs) Handles C1FlexGrid1.BeforeEdit
' 如果当前行是一个节点，则不允许编辑
Dim r As Row = C1FlexGrid1.Rows(C1FlexGrid1.Row)
If r.IsNode Then e.Cancel = True
End Sub
```

- C#

```
private void c1FlexGrid1_BeforeEdit( object sender, RowColEventArgs e)
{
//如果当前行是一个节点，则不允许编辑
Row r = c1FlexGrid1.Rows[c1FlexGrid1.Row];
if (r.IsNode ) e.Cancel = true;
}
```

6.2.5 步骤 5/5： 实现工具提示

在结束本教程前，我们将向大纲中添加工具提示。这个工具提示将把每一行中 UserData 属性中保存的上述 GetXMLData 的文本显示出来。 当用户将鼠标移到它的父节点时，工具提示将显示"Name" 节点的内容。这是非常有用的，因为当父节点折叠起来时，“Name”节点是不可见的。

1. 添加一个 **ToolTip** 控件到窗体上。
2. 添加下面的代码来处理表格上的 **MouseMove** 事件：

- Visual Basic

```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As
MouseEventArgs) Handles C1FlexGrid1.MouseMove
' 为当前单元格检查 ToolTip
Dim tip As String
If C1FlexGrid1.MouseCol = 0 And C1FlexGrid1.MouseRow > 0 Then
tip = C1FlexGrid1.Rows(C1FlexGrid1.MouseRow).UserData
' 如果与当前提示不同，则设置 ToolTip。
If tip <> ToolTip1.GetToolTip(C1FlexGrid1) Then
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End If
End If
End Sub
```

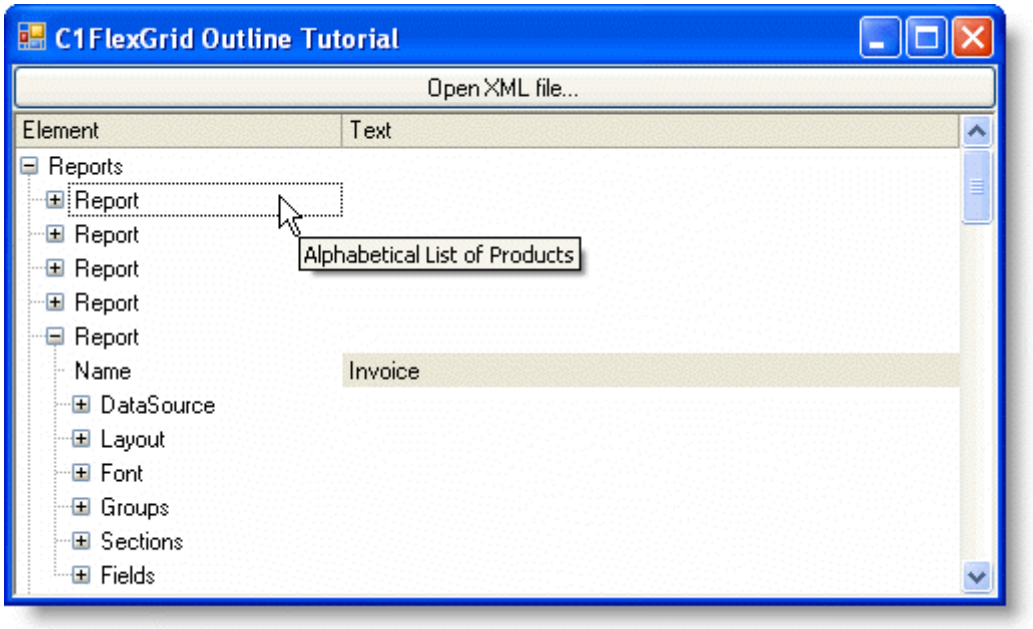
- C#

```
private void c1FlexGrid1_MouseMove( object sender, MouseEventArgs e)
{
//为当前单元格检查 ToolTip
string      tip;
if ( c1FlexGrid1.MouseCol == 0 && c1FlexGrid1.MouseRow > 0 )
{
tip = (string)c1FlexGrid1.Rows[c1FlexGrid1.MouseRow].UserData;
//如果与当前提示不同，则设置 ToolTip。
if ( tip != toolTip1.GetToolTip(c1FlexGrid1) )
{
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
}
}
```

运行程序，并遵守以下规定：

代码首先使用 MouseRow 属性和 MouseCol 属性来检查当前鼠标下的单元格。如果鼠标是在包含 ToolTip 文本行的第一列，则检索该行文本。否则，ToolTip 文本设置为 **Nothing。**

然后程序会比较当前和新的 ToolTip 文本，如有必要的话则通过调用 ToolTip 控件的 **SetToolTip** 方法来更新文本。



这样就结束了本教程。你可以在很多方面来扩展这个项目，包括将改动保存到 XML 文档中，添加、删除和移动节点，为不同类型的数据使用不同的样式等等。

6.3 数据分析教程

本教程结合了一些 C1FlexGrid 控件非常有用的功能来实现数据表的动态视图。这个应用程序是从一个简单的数据绑定表格开始，表格中包含了销售数据（来自 NorthWind 数据库），之后添加了一下功能：

- 动态布局（列顺序）
- 自动排序
- 合并单元格
- 自动合计
- 大纲视图

最终的程序看起来如下图所示。用户可以拖动前三列来分组数据，包括 salesperson、 country、 和 product name。当一列被拖动，最终合计会自动的重新计算并重建大纲树。

6.3.1 步骤 1/4： 为数据分析教程创建 C1FlexGrid 控件

创建一个新的项目，并在工具箱中点击 C1FlexGrid 图标来在窗体中添加一个 C1FlexGrid 控件，然后点击窗体并拖动控件直到控件有一个合适大小。

如果你没有在工具箱中找到 C1FlexGrid 控件，那么请在工具箱中右键单击然后选择“**选择项**...”。 然后在.Net 组件列表中查找 C1FlexGrid 控件并确保勾选上。 如果你在控件列表中无法找到控件，你也许需要重新安装控件包。

1. 在 **C1FlexGrid 事件**菜单中，点击 Dock 来将控件适用到父容器中。将表格的 **Dock** 属性设置为 **Fill** 以便表格填充整个窗体。

2. 双击窗体标题栏区域来打开代码窗口。在文件的最上端，添加下面的代码段：

- Visual Basic

```
Imports System.Data.OleDb
Imports System.ComponentModel
Imports C1.Win.C1FlexGrid
```

- C#

```
Using System.Data.OleDb;
Using System.ComponentModel;
Using C1.Win.C1FlexGrid;
```

这使得 C1FlexGrid 中定义的对象和 OleDb 程序集在整个项目都可见，并节省了很多需要敲的代码。

6.3.2 步骤 2/4： 初始化和填充表格

要设置表格并用我们想要分析的销售数据来填充表格，可以在设计器中或在代码中设置布局属性和样式属性，并使用 **GetDataSource** 方法来填充表格。

1. 在设计器中或在代码中设置布局属性和样式属性。

在设计器中：

在属性窗口中，设置如下属性：

属性	设置
AllowEditing	False
AllowSorting	None
AllowMerging	Nodes
ExtendLastCol	True
SelectionMode	Cell
Tree.Style	Simple
Tree.Column	1

- 通过在 **C1FlexGrid 任务**菜单选择样式打开 **C1FlexGrid 样式编辑器**。想要知道对于如何访问 **C1FlexGrid 样式编辑器**的更多细节，请查看访问 C1FlexGrid 样式编辑器章节（第 143 页）
- 从内置样式列表中选择 **Normal**。
- 设置 Border.Style 属性为 None，设置 Trimming 属性为 EllipsisCharacter。
- 从内置样式列表中选择 **Subtotal0**。
- 设置 BackColor 属性为 Gold，设置 ForeColor 属性为 Black。
- 为 Subtotal1 和 Subtotal2 设置如下属性：

Subtotal1	
BackColor	Khaki
ForeColor	Black
Subtotal2	
BackColor	LightGoldenrodYellow
ForeColor	Black

- 点击 OK 来关闭编辑器。

在代码中：

添加下面的代码到 **Form_Load** 事件中来设置表格的布局和样式：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
EventArgs) Handles MyBase.Load
' 初始化表格的布局/行为。
C1FlexGrid1.AllowEditing = False
C1FlexGrid1.AllowSorting = AllowSortingEnum.None
C1FlexGrid1.AllowMerging = AllowMergingEnum.Nodes
C1FlexGrid1.SelectionMode = SelectionModeEnum.Cell
C1FlexGrid1.ExtendLastCol = True
C1FlexGrid1.Cols(0).Width = C1FlexGrid1.Cols.DefaultSize / 4
C1FlexGrid1.Tree.Style = TreeStyleFlags.Simple
C1FlexGrid1.Tree.Column = 1
' 初始化表格样式。
C1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.None
C1FlexGrid1.Styles.Normal.Trimming = StringTrimming.EllipsisCharacter
Dim s As CellStyle = C1FlexGrid1.Styles(CellStyleEnum.GrandTotal)
s.BackColor = Color.Black
s.ForeColor = Color.White
s = C1FlexGrid1.Styles(CellStyleEnum.Subtotal0)
s.BackColor = Color.Gold
s.ForeColor = Color.Black
s = C1FlexGrid1.Styles(CellStyleEnum.Subtotal1)
s.BackColor = Color.Khaki
s.ForeColor = Color.Black
s = C1FlexGrid1.Styles(CellStyleEnum.Subtotal2)
s.BackColor = Color.LightGoldenrodYellow
s.ForeColor = Color.Black
End Sub
```

- C#

```
private void Form1_Load( System.object sender, EventArgs e)
{
//初始化表格的布局/行为。
c1FlexGrid1.AllowEditing = false;
c1FlexGrid1.AllowSorting = AllowSortingEnum.None;
c1FlexGrid1.AllowMerging = AllowMergingEnum.Nodes;
c1FlexGrid1.SelectionMode = SelectionModeEnum.Cell;
c1FlexGrid1.ExtendLastCol = true;
c1FlexGrid1.Cols[0].Width = c1FlexGrid1.Cols.DefaultSize / 4;
c1FlexGrid1.Tree.Style = TreeStyleFlags.Simple;
c1FlexGrid1.Tree.Column = 1;
//初始化表格样式。
c1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.None;
c1FlexGrid1.Styles.Normal.Trimming = StringTrimming.EllipsisCharacter;
CellStyle s = c1FlexGrid1.Styles[CellStyleEnum.GrandTotal];
s.BackColor = Color.Black;
s.ForeColor = Color.White;
s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal0];
s.BackColor = Color.Gold;
s.ForeColor = Color.Black;
s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal1];
s.BackColor = Color.Khaki;
s.ForeColor = Color.Black;
s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal2];
s.BackColor = Color.LightGoldenrodYellow;
s.ForeColor = Color.Black;
}
```

常规的一开始设置表格的布局和一些样式。

2. 通过添加下面的代码到 Form_Load 事件中来将 C1FlexGrid 绑定到数据源。

- Visual Basic

```
' 将 C1FlexGrid 绑定到数据源。
C1FlexGrid1.DataSource = GetDataSource()
```

- C#

```
//将 C1FlexGrid 绑定到数据源。
c1FlexGrid1.DataSource = GetDataSource();
```

下面例行列出通过结合 **GetDataSource** 方法来创建一个数据源。

3. 将最后三列的 AllowDragging 属性设置为 False，用来锁定最后三列位置。这么做的目的是为了防止用户在这些列中做数据分组（这些列中的值是由每一行的不同得来的）。这个属性可以在在设计器中或者在代码中设置：

在设计器中：

- 在表格中选择 **Column 4**。这将会打开 **Column 4** 的列任务菜单。
- 取消勾选 Allow Dragging 复选框。
- 为 Column 5 和 Column 6 重复做上面两步。

另外，这个 AllowDragging 属性同样可以在 C1FlexGrid 列编辑器中设置：

- 在 **C1FlexGrid 任务**菜单中选择设计器来打开 **C1FlexGrid 列编辑器**。想要知道对于如何访问 **C1FlexGrid 列编辑器**的更多细节，请查看访问 C1FlexGrid 列编辑器章节（第 143 页）。
- 在右侧窗格中选择 **Column 4**。
- 在左侧窗格中，设置 AllowDragging 属性为 False。
- 将 Column 5 和 Column 6 的 AllowDragging 属性设置为 False。
- 不要关闭编辑器。

在代码中：

将下面的代码添加到 **Form_Load** 事件中：

- Visual Basic

```
' 防止用户拖动最后三列。
C1FlexGrid1.Cols(4).AllowDragging = False
C1FlexGrid1.Cols(5).AllowDragging = False
C1FlexGrid1.Cols(6).AllowDragging = False
```

- C#

```
//防止用户拖动最后三列。
c1FlexGrid1.Cols[4].AllowDragging = false;
c1FlexGrid1.Cols[5].AllowDragging = false;
c1FlexGrid1.Cols[6].AllowDragging = false;
```

4. 将 Sales Amount 列的 Format 属性设置一下，以便总和可以显示为货币样式。

这个操作可以在设计器中或者在代码中做到：

在设计器中：

- 在表格中选择 Column 6。
- 在 Format String 框中点击省略号按钮来打开 Format String 窗口。
- 在 Format type 中选择 Currency（货币）。
- 点击 OK 来关闭 Format String 窗口。

另外，Format 属性也可以通过使用 **C1FlexGrid 列编辑器**来设置：

- 在 **C1FlexGrid 列编辑器**中，在右侧窗格中选择 Column 6。
- 在左侧窗格中，点击 **Format** 属性旁边的**省略号**按钮来打开 **Format String** 对话框。
- 在 Format type 中选择 **Currency（货币）**。
- 点击 OK 来关闭 Format String 窗口。
- 点击 OK 来关闭编辑器。

在代码中：

- Visual Basic

```
' 在 Sales Amount 列显示货币值。
C1FlexGrid1.Cols(6).Format = "c"
```

- C#

```
//在 Sales Amount 列显示货币值。
c1FlexGrid1.Cols[6].Format = "c";
```

-

5. **GetDataSource** 方法创建表格中显示的数据表。该例程是非常基本的，除了检索数据的 SQL 语句。大多数人不手动编写 SQL 语句，一般使用虚拟设计

器如 Visual Studio 或者 Microsoft Access 来做。

将下面的代码添加到窗体中。请注意，您可能必须稍微更改一下连接字符串，因为它引用了 NorthWind 数据库，该数据库文件可能是在您系统中不同的文件夹中：

- Visual Basic

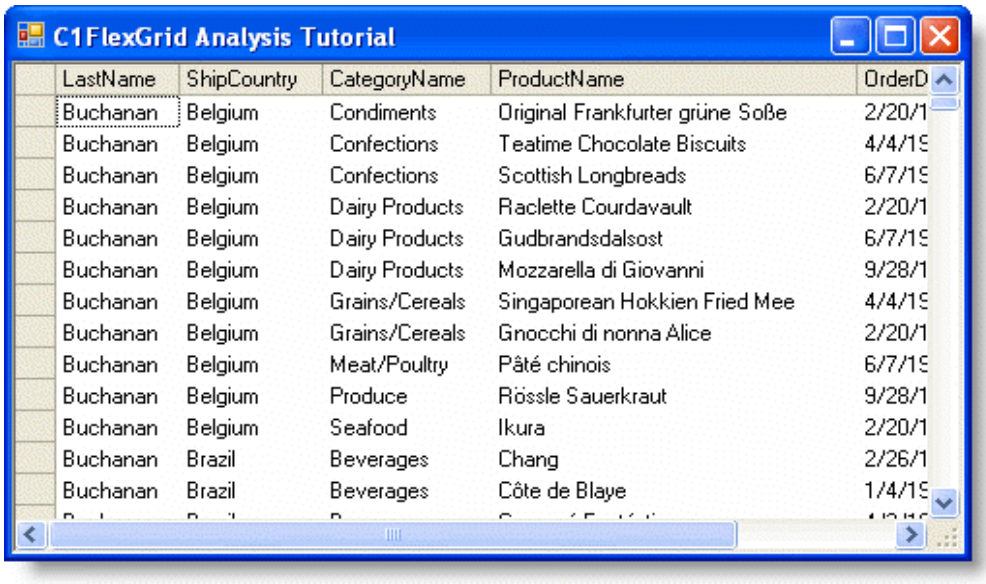
```
    Private Function GetDataSource() As DataTable
    ' 创建连接字符串。
    Dim conn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\Program Files\ComponentOne Studio.NET
2.0\Common\Nwind.mdb"
    ' 创建 SQL 语句。
    Dim rs As String = _
    "SELECT Employees.LastName,Orders.ShipCountry," & _
    "Categories.CategoryName,Products.ProductName,Orders.OrderDate," & _
    "[Quantity]*[Products].[UnitPrice] AS [Sale Amount] " & _
    "FROM (Categories INNER JOIN Products " & _
    "ON Categories.CategoryID = Products.CategoryID) " & _
    "INNER JOIN ((Employees INNER JOIN Orders " & _
    "ON Employees.EmployeeID = Orders.EmployeeID) " &
      _
    "INNER JOIN [Order Details] " & _
    "ON Orders.OrderID = [Order Details].OrderID) " &
      _
    "ON Products.ProductID = [Order Details].ProductID " & _
    "ORDER BY Employees.LastName,Orders.ShipCountry," & _
"Categories.CategoryName;"
    ' 检索数据并放到 Dataset 中。
    Dim da As OleDbDataAdapter = New OleDbDataAdapter(rs, conn)
    Dim ds As DataSet = New DataSet()
    da.Fill(ds)
    ' 返回数据表
    GetDataSource = ds.Tables(0)
    End Function
```

- C#

```
private DataTable GetDataSource()
{
//创建连接字符串。
string conn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
"Data Source=C:\\Program Files\\ComponentOne Studio.NET
2.0\\Common\\Nwind.mdb";
//创建 SQL 语句。
string rs =
"SELECT Employees.LastName,Orders.ShipCountry," +
"Categories.CategoryName,Products.ProductName,Orders.OrderDate," +
"[Quantity]*[Products].[UnitPrice] AS [Sale Amount] " +
"FROM (Categories INNER JOIN Products " +
"ON Categories.CategoryID = Products.CategoryID) " +
"INNER JOIN ((Employees INNER JOIN Orders " +
"ON Employees.EmployeeID = Orders.EmployeeID) " +
"INNER JOIN [Order Details] " +
"ON Orders.OrderID = [Order Details].OrderID) " +
"ON Products.ProductID = [Order Details].ProductID " +
"ORDER BY Employees.LastName,Orders.ShipCountry," +
"Categories.CategoryName;";
//检索数据并放到 Dataset 中。
OleDbDataAdapter da = new OleDbDataAdapter(rs, conn);
DataSet ds = new DataSet();
da.Fill(ds);
//返回数据表
return ds.Tables[0];
}
```

运行程序，并遵守以下规定：

你会看到一个普通的有前瞻性的表格，它可以让你移动列，并浏览数据。但是，这些数据没有一个明确的结构，这个表包含了几千条记录，所以想要知道得到的数据意味着什么是相当困难的。



6.3.3 步骤 3/4： 允许自动排序

在整理数据的第一步是排序。此外，我们想，当用户重新安排这些列时，让这些数据能够自动排序。

在用户重新安排这些列之后，C1FlexGrid 控件触发 AfterDragColumn 事件。我们将添加一个事件处理程序，以基础数据表中的数据进行排序。（如果表格处在非绑定模式，我们可以使用 Sort 方法来做到这一点。）

向窗体中添加下面的代码，用来在用户拖动列的时候排序数据记录和重建合计：

- Visual Basic

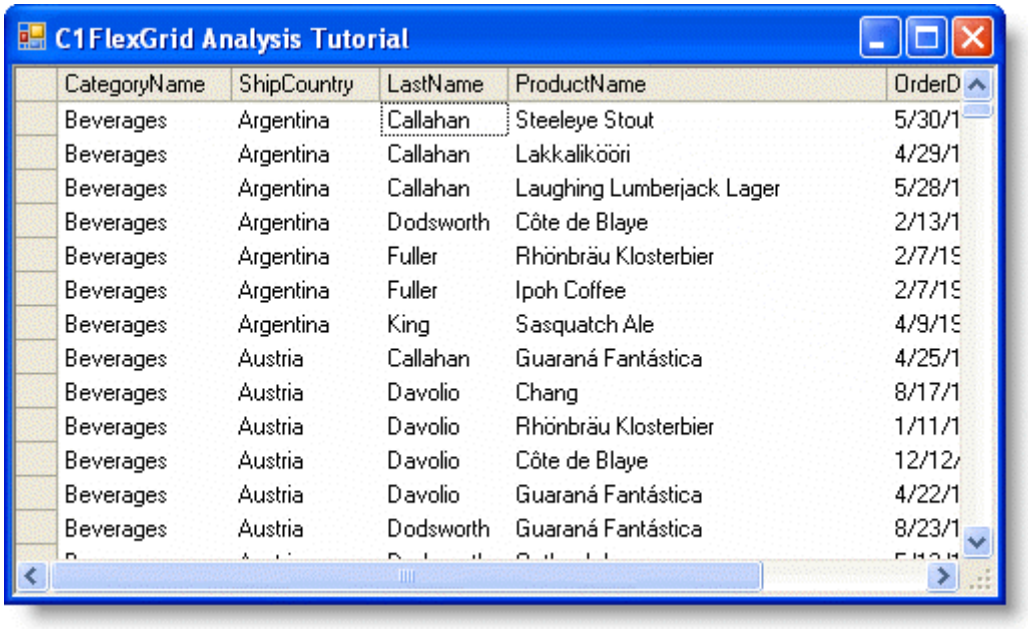
```
Private Sub C1FlexGrid1_AfterDragColumn(ByVal sender As Object, ByVal e
As
DragRowColEventArgs) Handles C1FlexGrid1.AfterDragColumn
' 当用户拖动列的时候排序数据。
' 这将导致数据刷新，移除所有合计和触发 AfterDataRefresh 事件，并重
建合计。
Dim sort As String = C1FlexGrid1.Cols(1).Name & ", " & _
C1FlexGrid1.Cols(2).Name & ", " & _
C1FlexGrid1.Cols(3).Name
Dim dt As DataTable = C1FlexGrid1.DataSource
dt.DefaultView.Sort = sort
End Sub
```

- C#

```
private void c1FlexGrid1_AfterDragColumn( object sender,
DragRowColEventArgs e)
{
//当用户拖动列的时候排序数据。
//这将导致数据刷新，移除所有合计和触发 AfterDataRefresh 事件，并重
建合计。
string sort = c1FlexGrid1.Cols[1].Name + ", " +
c1FlexGrid1.Cols[2].Name + ", " +
c1FlexGrid1.Cols[3].Name;
DataTable dt = (DataTable)c1FlexGrid1.DataSource;
dt.DefaultView.Sort = sort;
}
```

运行程序，并遵守以下规定：

运行该项目，并尝试通过拖动前三列的标题来重新对它们排序。每当你移动一列，数据将自动进行排序，这将使得它更加容易理解。



在下一个步骤中，我们将添加合计和一个大纲树。

6.3.4 步骤 4/4： 添加合计和大纲树

当表格在绑定模式下使用时，任何数据源的变化都会导致表格触发 AfterDataRefresh 事件。这个事件是向表格中插入分类汇总和创建大纲树的理想代码场所。添加下面的 AfterDataRefresh 事件代码处理到窗体中：

- Visual Basic

```
Private Sub C1FlexGrid1_AfterDataRefresh(ByVal sender As Object, ByVal e
As ListChangedEventArgs) Handles C1FlexGrid1.AfterDataRefresh
' Sale Amount 的总和。
Dim totalOn As Integer = C1FlexGrid1.Cols("Sale Amount").SafeIndex
Dim caption As String = "Total for {0}"
' 计算三层的总和。
C1FlexGrid1.Subtotal(AggregateEnum.Sum, 0, 1, totalOn, caption)
C1FlexGrid1.Subtotal(AggregateEnum.Sum, 1, 2, totalOn, caption)
C1FlexGrid1.Subtotal(AggregateEnum.Sum, 2, 3, totalOn, caption)
' 将大纲树第二级折叠起来。
C1FlexGrid1.Tree.Show(2)
' 表格自动调整大侠以适应大纲树。
C1FlexGrid1.AutoSizeCols(1, 1, 1000, 3, 30, AutoSizeFlags.IgnoreHidden)
End Sub
```

- C#

```
private void c1FlexGrid1_AfterDataRefresh( object sender,
ListChangedEventArgs e)
{
// Sale Amount 的总和。
int totalOn = c1FlexGrid1.Cols["Sale Amount"].SafeIndex;
string      caption = "Total for {0}";
// 计算三层的总和。
c1FlexGrid1.Subtotal(AggregateEnum.Sum, 0, 1, totalOn, caption);
c1FlexGrid1.Subtotal(AggregateEnum.Sum, 1, 2, totalOn, caption);
c1FlexGrid1.Subtotal(AggregateEnum.Sum, 2, 3, totalOn, caption);
//将大纲树第二级折叠起来。
c1FlexGrid1.Tree.Show(2);
//表格自动调整大侠以适应大纲树。
c1FlexGrid1.AutoSizeCols(1, 1, 1000, 3, 30, AutoSizeFlags.IgnoreHidden);
}
```

运行程序，并遵守以下规定：

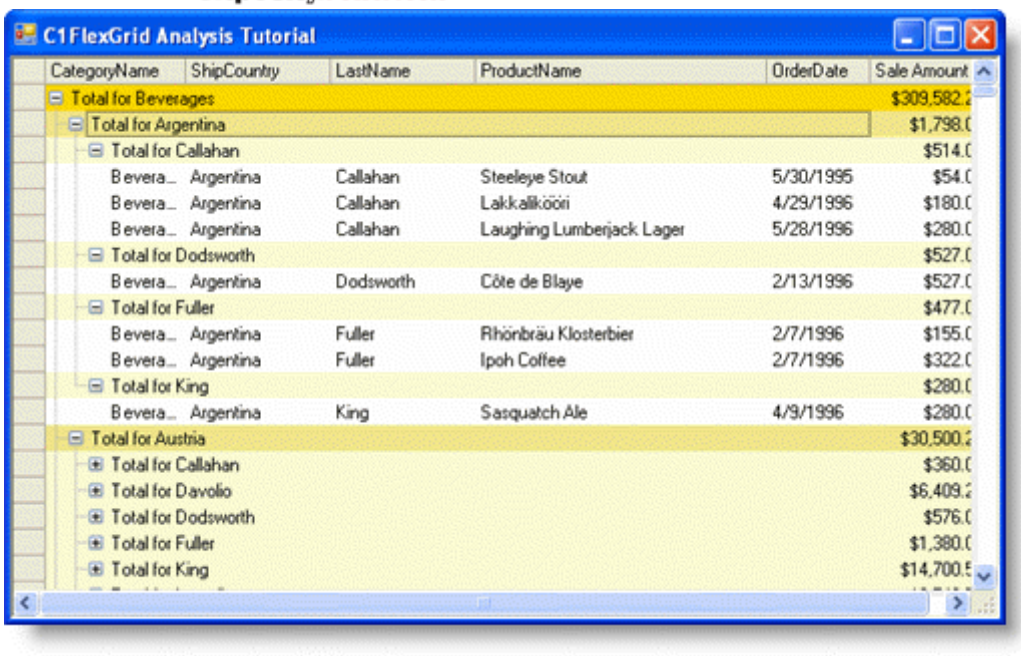
代码首先获得 Sale Amount 列的索引，本教程中，这个索引始终是相同的（Column 6）。按照索引通常比硬编码要好，但是，因为如果有人在 SQL 语句中添加了多个字段，该索引将会发生改变。

代码调用了 C1FlexGrid.的 Subtotal 方法来对数据进行分组并为合计添加新行。这个新行将会自动配置成大纲树的节点（它们的 IsNode 被设置为 **True**），以便合计和折叠起来。

试着拖动一些列，你可以清楚的看到 country、product category 或者 salesperson 的合计。当然你如果想看到更多细节的话，也可以展开树的节点。

注意，表格是可以编辑的，改变 Sale Amount 列的值将会再次触发 AfterDataRefresh 事件，并且合计也会自动的更新。

这样就结束了本教程。



7. FlexGrid For WinForms 中基于任务的帮助

假定您熟悉 Visual Studio .NET 下基于任务的帮助编程，并知道如何使用一般的绑定与非绑定控件。如果你对于 **ComponentOne FlexGrid for WinForms** 控件产品来说是一个新手的话，请先查看 FlexGrid for WinForms 教程章节。

每个主题中，通过使用 **ComponentOne FlexGrid for WinForms** 控件，来为具体任务提供了一个解决方案。按照帮助中所述的步骤实现之后，您将能够创建项目来展示各种 **C1FlexGrid** 功能。

每个基于任务的帮助主题还假定您已经知道如何创建一个新的.NET 工程。

7.1 访问 C1FlexGrid 编辑器

C1FlexGrid 编辑器可以通过 **C1FlexGrid 任务菜单**、上下文菜单或者属性窗口来访问。有两个 C1FlexGrid 编辑器，一个是 **C1FlexGrid 列编辑器** ，另一个是 **C1FlexGrid 样式编辑器**，其中样式编辑器可以允许你在设计时控制 C1FlexGrid 的布局和外观。如果要对列进行重新排序、调整列宽、设置列属性、插入或删除一列，请使用 **C1FlexGrid 列编辑器。** 要更改现有的样式或者添加自定义样式，以便将它们指定到单元格、行、列上，请使用 **C1FlexGrid 样式编辑器。**

7.1.1 访问 C1FlexGrid 列编辑器

要访问 **C1FlexGrid 列编辑器**，请使用 **C1FlexGrid 任务菜单**、上下文菜单或者属性窗口。更多使用 **C1FlexGrid 列编辑**编辑列的信息，请查看 C1FlexGrid 列编辑器章节。

C1FlexGrid 任务菜单

单击 C1FlexGrid 右上角的智能标记（ ）来打开 **C1FlexGrid 任务菜单**，之后选择**设计器**。

上下文菜单

在窗体上右键单击并在上下文菜单中选择**设计器**。

属性窗口

在属性窗口中，点击 **Cols** 属性旁边的省略号按钮。

7.1.2 访问 C1FlexGrid 样式编辑器

要访问 **C1FlexGrid 样式编辑器**，请使用 **C1FlexGrid 任务菜单**、上下文菜单或者属性窗口。想要了解更多关于使用 **C1FlexGrid 样式编辑器**来定制单元格外观样式的信息，请查看 C1FlexGrid 样式编辑器 章节（第 27 页）。

C1FlexGrid 任务菜单

单击 C1FlexGrid 右上角的智能标记（ ）来打开 **C1FlexGrid 任务菜单**，之后选择**样式**。

上下文菜单

在窗体上右键单击并在上下文菜单中选择**样式**。

属性窗口

在属性窗口中，点击 **Styles** 属性旁边的省略号按钮。

7.2 向单元格中添加图片和文本

要向单元格中添加图片和文本，请使用 SetData 和 SetCellImage 方法。添加如下代码到 **Form_Load** 事件中去：

1. 调整单元格以适应图像的高度和宽度属性。

- Visual Basic

```
Me.C1FlexGrid1.Rows(1).Height = 90
Me.C1FlexGrid1.Cols(1).Width = 150
```

- C#

```
this.c1FlexGrid1.Rows[1].Height = 90;
this.c1FlexGrid1.Cols[1].Width = 150;
Me.C1FlexGrid1.SetCellImage(1, 1, Image.FromFile("c:\c1logo.bmp"))
```

2. 使用 SetCellImage 方法来添加图片：

- Visual Basic

```
Me.C1FlexGrid1.SetCellImage(1, 1, Image.FromFile("c:\c1logo.bmp"))
```

- C#

```
this.c1FlexGrid1.SetCellImage(1, 1, Image.FromFile(@"c:\c1logo.bmp"));
```

3. 使用 SetData 方法来添加文本：

- Visual Basic

```
Me.C1FlexGrid1.SetData(1, 1, "ComponentOne")
```

- C#

```
this.c1FlexGrid1.SetData(1, 1, "ComponentOne");
```

4. 设置图片的对齐方式为 **CenterTop**，设置文本的对齐方式为 **CenterBottom**：

- Visual Basic

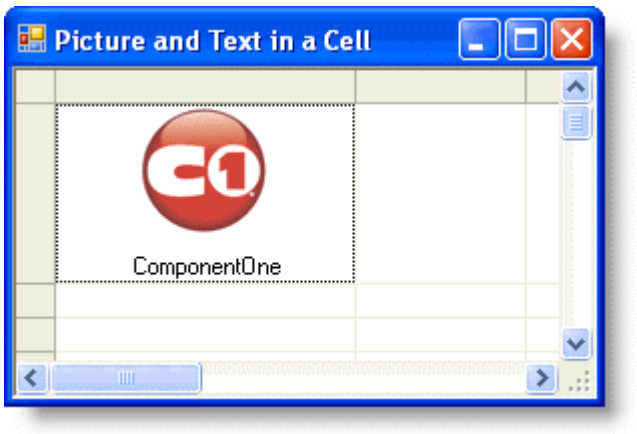
```
Me.C1FlexGrid1.Styles.Normal.ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.CenterTop
Me.C1FlexGrid1.Styles.Normal.TextAlign =
C1.Win.C1FlexGrid.TextAlignEnum.CenterBottom
```

- C#

```
this.c1FlexGrid1.Styles.Normal.ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.CenterTop;
this.c1FlexGrid1.Styles.Normal.TextAlign =
C1.Win.C1FlexGrid.TextAlignEnum.CenterBottom;
```

本主题演示如下：

您的表格将如下面所示，一张图片和文本内容在同一个单元格内。



注意： 要将文本放置到图片的上方，将文本对齐方式改为 **CenterTop**，图片对齐方式改为 **CenterBottom**。

7.3 向固定列中添加行号

向固定列中添加行号，比如 Microsoft Excel，请使用 OwnerDrawCell 时间来在固定列中画出数字并左对齐。

1. 添加下面的代码到 **Form_Load** 事件中，触发 OwnerDrawCell 事件：

- Visual Basic

```
    Me.C1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
```

- C#

```
    this.c1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
```

2. 添加 OwnerDrawCell 事件：

- Visual Basic

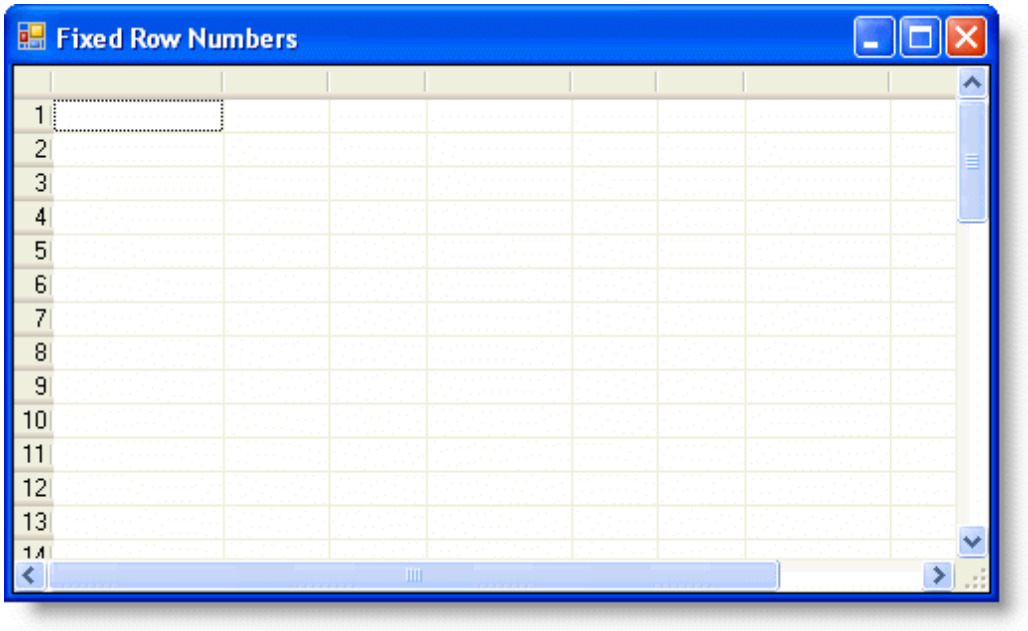
```
    Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e
    As C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles
C1FlexGrid1.OwnerDrawCell
    If e.Row >= Me.C1FlexGrid1.Rows.Fixed And e.Col =
Me.C1FlexGrid1.Cols.Fixed - 1 Then
    Dim rowNumber As Integer = e.Row - Me.C1FlexGrid1.Rows.Fixed + 1
    e.Text = rowNumber.ToString()
    End If
    End Sub
```

- C#

```
private void c1FlexGrid1_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
if ((e.Row >= this.c1FlexGrid1.Rows.Fixed) & (e.Col ==
(this.c1FlexGrid1.Cols.Fixed - 1)))
{
e.Text = ((e.Row - this.c1FlexGrid1.Rows.Fixed) + 1).ToString();
}
}
```

本主题演示如下：

行号出现在第一列，它是固定的并左对齐，就像 Microsoft Excel 一样。



7.4 向标题行添加三维文字

要向表的标题行添加三维文字，将文字的 TextEffect 属性设置为 **Raised**，意思是文字有阴影，向右边偏移一个像素并位于文字下方，或者将 TextEffect 属性设置为 **Inset**，意思是阴影向文字左边偏移一个像素并位于文字上方。将 **TextEffect** 属性设置为 **Flat** 将没有任何效果。

1. 创建一个新的样式名为 3DText。

在设计器中：

- 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 **C1FlexGrid 样式编辑器**的细节，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
- 点击 Add 来创建一个新的样式。
- 双击 CustomStyle1，将它改名为 3DText，完成后请按回车键。
- 不要退出 C1FlexGrid 样式编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
Dim tdt As C1.Win.C1FlexGrid.CellStyle
tdt = Me.C1FlexGrid1.Styles.Add("3DText")
```

- C#

```
C1.Win.C1FlexGrid.CellStyle tdt = this.c1FlexGrid1.Styles.Add("3Dtext");
```

2. 设置 TextEffect 属性为 Raised。

在设计器中：

在右侧窗格中找到 TextEffect 属性并将它设置为 Raised。

在代码中：

将下面的代码添加到 Form_Load 事件中：

- Visual Basic

```
tdt.TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Raised
```

- C#

```
tdt.TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Raised;
```

3. 应用该样式到标题行，将以下代码添加到 Form_Load 事件中：

- Visual Basic

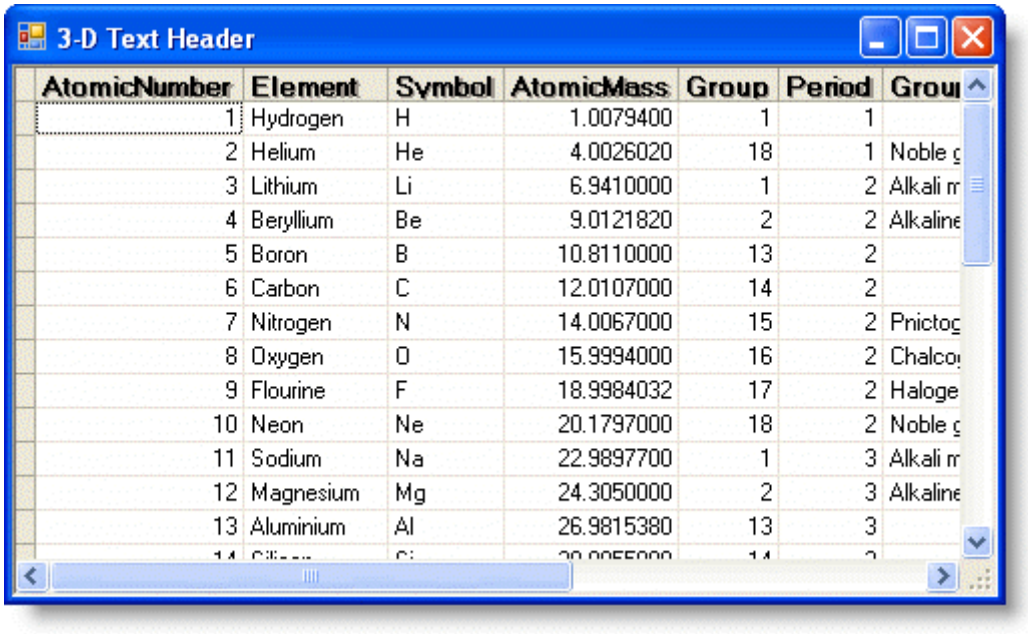
```
Me.C1FlexGrid1.Rows(0).Style = Me.C1FlexGrid1.Styles("3DText")
```

- C#

```
this.c1FlexGrid1.Rows[0].Style = this.c1FlexGrid1.Styles["3DText"];
```

本主题演示如下：

表中应该有一个类似下面图片的标题行，浮起的文字效果。



7.4.1 使用内置的样式向标题行添加三维文字

TextEffect 属性可以使用 **C1FlexGrid** 的内置样式添加一个三维文本效果。要使用样式添加一个三维文本标题行，将 TextEffect 属性设置为内置的 **Raised** 样式，意思是文字有阴影，向右边偏移一个像素并位于文字下方，或者将 TextEffect 属性设置为 **Inset**，意思是阴影向文字左边偏移一个像素并位于文字上方。将 **TextEffect** 属性设置为 **Flat** 将没有任何效果。这个属性可以在设计器中或在代码中设置：

在设计器中：

1. 打开 C1FlexGrid 样式编辑器。想要了解更多关于如何访问 C1FlexGrid 样式编辑器的细节，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
2. 在内置样式中选择 Fixed。
3. 在右侧窗格中，找到 **TextEffect** 属性并设置为 **Raised**。
4. 点击 OK 来关闭编辑器。

在代码中：

将下面代码添加到 **Form_Load** 事件中，设置固定列的 **TextEffect** 属性样式。

- Visual Basic

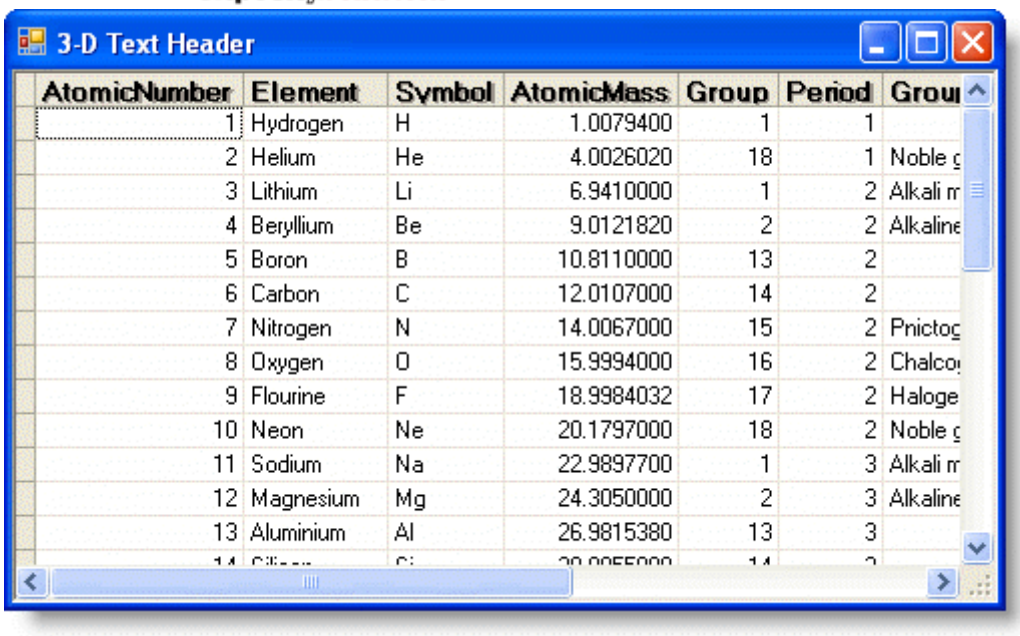
```
Me.C1FlexGrid1.Styles("Fixed").TextEffect =
C1.Win.C1FlexGrid.TextEffectEnum.Raised
```

- C#

```
this.c1FlexGrid1.Styles["Fixed"].TextEffect =
C1.Win.C1FlexGrid.TextEffectEnum.Raised;
```

本主题演示如下：

通过设置固定列样式的 **TextEffect** 属性，只有表中固定的单元格会拥有三维效果。**TextEffect** 属性可以被设为任何内置样式。查看 CellStyleEnum 枚举集来了解内置样式列表和描述。


3-D Text Header
AtomicNumber Element Symbol AtomicMass Group Period Grou
1 Hydrogen H 1.0079400 1 1
2 Helium He 4.0026020 18 1 Noble g
3 Lithium Li 6.9410000 1 2 Alkali m
4 Beryllium Be 9.0121820 2 2 Alkaline
5 Boron B 10.8110000 13 2
6 Carbon C 12.0107000 14 2
7 Nitrogen N 14.0067000 15 2 Pnictog
8 Oxygen O 15.9994000 16 2 Chalco
9 Flourine F 18.9984032 17 2 Haloge
10 Neon Ne 20.1797000 18 2 Noble g
11 Sodium Na 22.9897700 1 3 Alkali m
12 Magnesium Mg 24.3050000 2 3 Alkaline
13 Aluminium Al 26.9815380 13 3

7.5 添加 ToolTips 用来显示 UserData

要添加 ToolTips 用来显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 事件来拿到 **UserData** 属性并将它显示在 ToolTip 控件中。ToolTips 可以被设置到一列、一行、一个单元格范围、单元格样式或单个的单元格。

7.5.1 列上的 UserData ToolTips

要添加 ToolTips 用来在一列上显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 事件来拿到 **UserData** 属性并将它显示在 ToolTip 控件中。

1. 在工具箱中找到 **ToolTip** 控件，并将它添加到窗体中。
2. 在 Form_Load 事件中，将 AtomicMass 列的 UserData 属性设置为：

- Visual Basic

```
Me.C1FlexGrid1.Cols("AtomicMass").UserData = "in atomic mass units (u)"
```

- C#

```
this.c1FlexGrid1.Cols["AtomicMass"].UserData = "in atomic mass units (u)";
```

3. 添加下面的代码到 MouseMove 事件中：

- Visual Basic

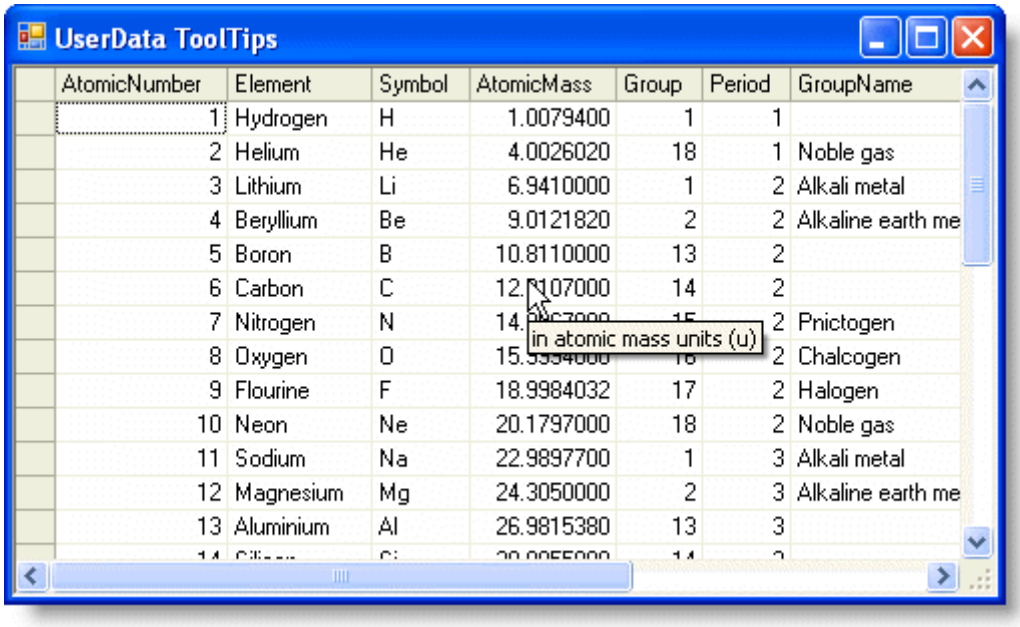
```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1FlexGrid1.MouseMove
Dim tip As String
tip = C1FlexGrid1.Cols(C1FlexGrid1.MouseCol).UserData
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End Sub
```

- C#

```
private void c1FlexGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
string tip;
tip = (string)c1FlexGrid1.Cols[c1FlexGrid1.MouseCol].UserData;
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
```

本主题演示如下：

ToolTip 将会出现在整个 AtomicMass 列中。



7.5.2 单元格区域中的 UserData ToolTips

要添加 ToolTips 用来在单元格范围中显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 来拿到 CellRange.UserData 属性并将它显示在 ToolTip 控件中。

1. 在工具箱中找到 ToolTip 控件，并将它添加到窗体中。
2. 在 Form_Load 事件中，将包含了一组非金属元素的单元格区域的 UserData 属性设置为：

- Visual Basic

```
' 取得非金属元素的单元格区域
Dim rng As C1.Win.C1FlexGrid.CellRange = Me.C1FlexGrid1.GetCellRange(6, 1, 10, 10)
' 将背景色改变一下，这样便于看到设置的单元格区域
rng.StyleNew.BackColor = Color.AliceBlue
' 为这一区域设置 UserData
rng.UserData = "Non-Metallic"
```

- C#

```
//取得非金属元素的单元格区域
C1.Win.C1FlexGrid.CellRange rng = this.c1FlexGrid1.GetCellRange(6, 1,
10, 10);
//将背景色改变一下，这样便于看到设置的单元格区域
rng.StyleNew.BackColor = Color.AliceBlue;
//为这一区域设置 UserData
rng.UserData = "Non-Metallic";
```

3. 将下面的代码添加到 MouseMove 事件中去：

- Visual Basic

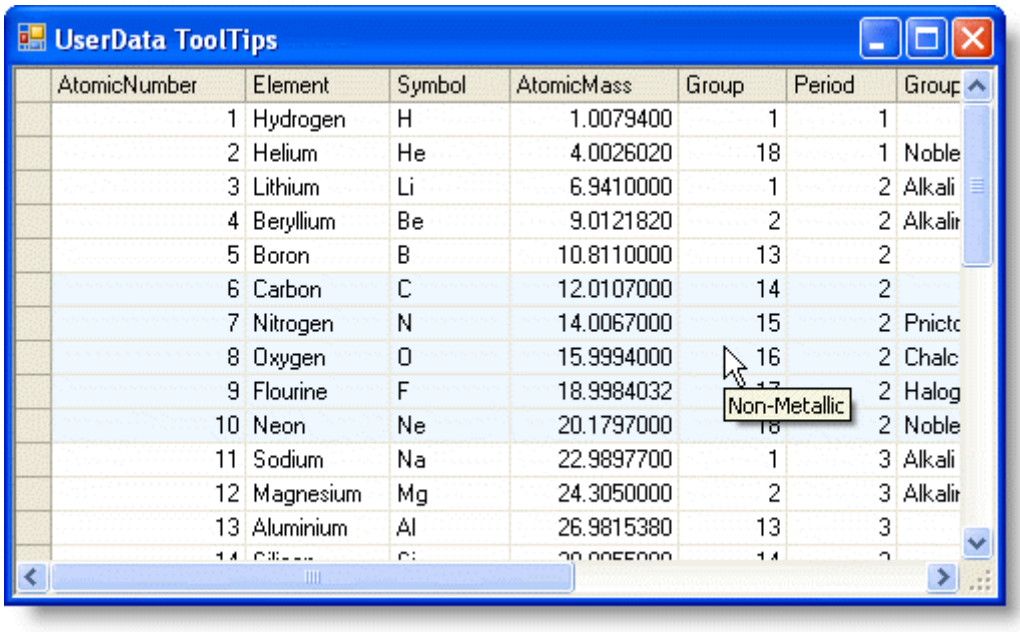
```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1FlexGrid1.MouseMove
Dim tip As String
tip = Me.C1FlexGrid1.GetUserData(C1FlexGrid1.MouseRow,
C1FlexGrid1.MouseCol)
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End Sub
```

- C#

```
private void c1FlexGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
string tip;
tip = (string)this.c1FlexGrid1.GetUserData(c1FlexGrid1.MouseRow,
c1FlexGrid1.MouseCol);
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
```

本主题演示如下：

ToolTip 将会出现在整个单元格区域中。



7.5.3 单元格样式中的 UserData ToolTips

要添加 ToolTips 用来在单元格样式中显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 事件来拿到 CellStyle.UserData 属性并将它显示在 ToolTip 控件中。

1. 在工具箱中找到 ToolTip 控件，并将它添加到窗体中。
2. 在 **Form_Load** 事件中，为列和行分别创建一个自定义的 CellStyle，然后将这两个样式指定到表格中的一列和一行上。想要了解关于如何创建 CellStyles 的信息，请查看为列和行设置背景色。

- Visual Basic

```
' 为列创建一个 CellStyle 。
Dim cc As C1.Win.C1FlexGrid.CellStyle
cc = Me.C1FlexGrid1.Styles.Add("ColumnColor")
cc.BackColor = Color.Cornsilk
' 将样式名称添加到 UserData。
cc.UserData = "ColumnColor Style"
' 为行创建一个 CellStyle 。
Dim rs As C1.Win.C1FlexGrid.CellStyle
rs = Me.C1FlexGrid1.Styles.Add("RowColor")
rs.BackColor = Color.PowderBlue
' 将样式名称添加到 UserData。
rs.UserData = "RowColor Style"
'  将创建的列样式指定到一列上。
Me.C1FlexGrid1.Cols(2).Style = Me.C1FlexGrid1.Styles("ColumnColor")
' 将创建的行样式指定到一行上。
Me.C1FlexGrid1.Rows(8).Style = Me.C1FlexGrid1.Styles("RowColor")
```

- C#

```
//为列创建一个 CellStyle 。
C1.Win.C1FlexGrid.CellStyle cc;
cc = this.c1FlexGrid1.Styles.Add("ColumnColor");
cc.BackColor = Color.Cornsilk;
//将样式名称添加到 UserData。
cc.UserData = "ColumnColor Style";
//为行创建一个 CellStyle 。
C1.Win.C1FlexGrid.CellStyle rs;
rs = this.c1FlexGrid1.Styles.Add("RowColor");
rs.BackColor = Color.PowderBlue;
//将样式名称添加到 UserData。
rs.UserData = "RowColor Style";
//将创建的列样式指定到一列上。
this.c1FlexGrid1.Cols[2].Style =
this.c1FlexGrid1.Styles["ColumnColor"];
//将创建的行样式指定到一行上。
this.c1FlexGrid1.Rows[8].Style = this.c1FlexGrid1.Styles["RowColor"];
```

3. 将下面的代码添加到 MouseMove 事件中：

- Visual Basic

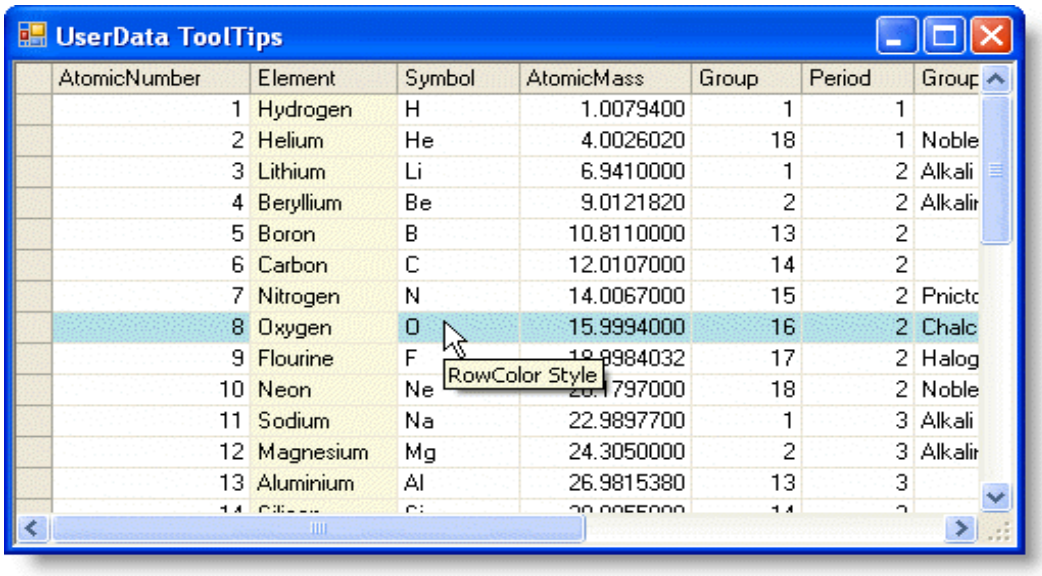
```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1FlexGrid1.MouseMove
Dim tip As String
If C1FlexGrid1.MouseCol <> C1FlexGrid1.Cols.Fixed - 1 And
C1FlexGrid1.MouseRow <> C1FlexGrid1.Rows.Fixed - 1 Then
tip = C1FlexGrid1.Cols(C1FlexGrid1.MouseCol).Style.UserData
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End If
End Sub
```

- C#

```
private void c1FlexGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
string tip;
if (c1FlexGrid1.MouseCol != c1FlexGrid1.Cols.Fixed - 1 &
c1FlexGrid1.MouseRow != c1FlexGrid1.Rows.Fixed - 1)
{
tip = (string)c1FlexGrid1.Cols[c1FlexGrid1.MouseCol].Style.UserData;
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
}
```

本主题演示如下：

ToolTip 将会出现在拥有 CellStyle 的范围内。



7.5.4 行上的 UserData ToolTips

要添加 ToolTips 用来在一行上显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 事件来拿到 **UserData** 属性并将它显示在 ToolTip 控件中。

1. 在工具箱中找到 **ToolTip** 控件，并将它添加到窗体中。
2. 在 Form_Load 事件中，将包含 Oxygen 行的 UserData 属性设置为：

- Visual Basic

```
Me.C1FlexGrid1.Rows(8).UserData = "Oxygen makes up approximately 1/5 of the earth's atmosphere."
```

- C#

```
this.c1FlexGrid1.Rows[8].UserData = "Oxygen makes up approximately 1/5 of the earth's atmosphere.";
```

3. 将下面的代码添加到 MouseMove 事件中：

- Visual Basic

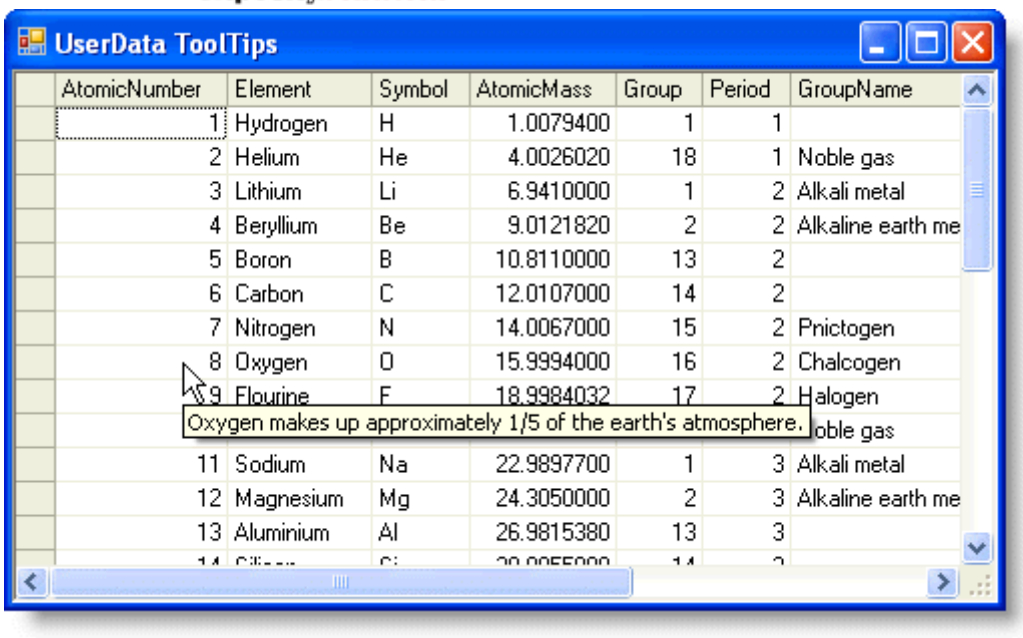
```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles C1FlexGrid1.MouseMove
Dim tip As String
tip = C1FlexGrid1.Rows(C1FlexGrid1.MouseRow).UserData
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End Sub
```

- C#

```
private void c1FlexGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
string tip;
tip = (string)c1FlexGrid1.Rows[c1FlexGrid1.MouseRow].UserData;
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
```

本主题演示如下：

ToolTip 将会出现在整行。



7.5.5 单一单元格上的 UserData ToolTips

要添加 ToolTips 用来在一个单元格上显示 **UserData**，请使用 **C1FlexGrid** 的 **MouseMove** 事件来拿到 **UserData** 并将它显示在 ToolTip 控件中。

1. 在工具箱中找到 ToolTip 控件，并将它添加到窗体中。
2. 将下面的 SetUserData 方法添加到 Form_Load 事件中：

- Visual Basic

```
Me.C1FlexGrid1.SetUserData(1, "Element", "Hydrogen is a highly flammable gas.")
```

- C#

```
this.c1FlexGrid1.SetUserData(1, "Element", "Hydrogen is a highly flammable gas.");
```

3. 将下面的代码添加到 **MouseMove** 事件中：

- Visual Basic

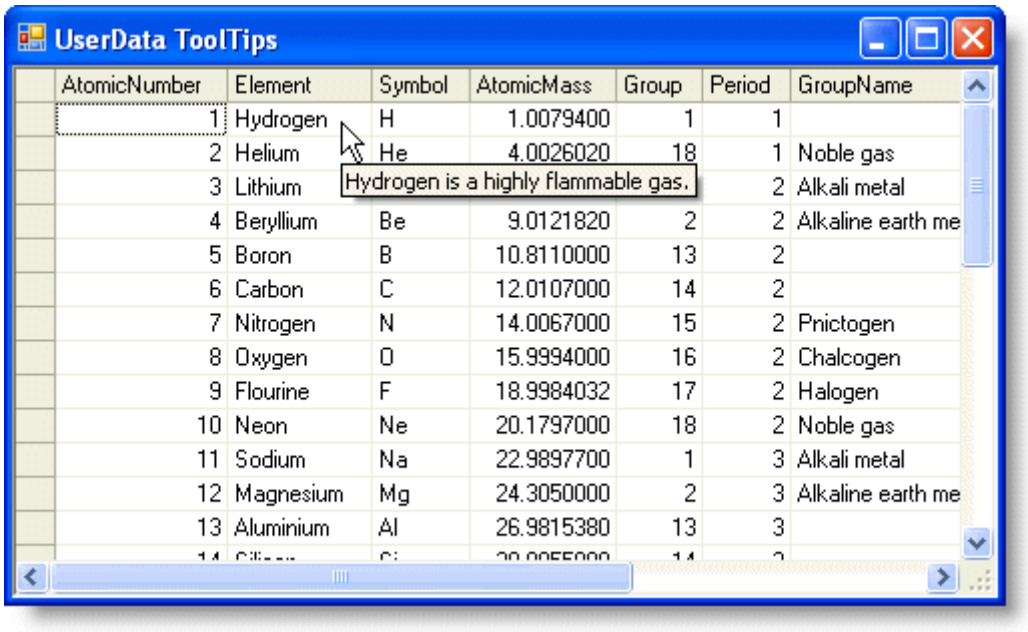
```
Private Sub C1FlexGrid1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1FlexGrid1.MouseMove
Dim tip As String
tip = Me.C1FlexGrid1.GetUserData(C1FlexGrid1.MouseRow,
C1FlexGrid1.MouseCol)
ToolTip1.SetToolTip(C1FlexGrid1, tip)
End Sub
```

- C#

```
private void c1FlexGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
string tip;
tip = (string)this.c1FlexGrid1.GetUserData(c1FlexGrid1.MouseRow,
c1FlexGrid1.MouseCol);
toolTip1.SetToolTip(c1FlexGrid1, tip);
}
```

本主题演示如下：

当鼠标移动到 Element 列下的第一行上面时，关于 Hydrogen 的信息就会出现。



7.6 为一个单元格区域应用渐变背景色

想要为一个单元格区域添加渐变背景色，请使用 OwnerDrawCell 事件来创

建自定义绘制单元格。

1. 创建一个 LinearGradient 画刷并定义一个单元格区域，将下面的代码添加到窗体的类中去：

- Visual Basic

```
Dim GradientStyleBrush As System.Drawing.Drawing2D.LinearGradientBrush
Dim rng As C1.Win.C1FlexGrid.CellRange
```

- C#

```
System.Drawing.Drawing2D.LinearGradientBrush GradientStyleBrush;
C1.Win.C1FlexGrid.CellRange rng;
```

2. 通过添加下面的代码到 Form_Load 事件中，可以将 C1FlexGrid 的 DrawMode 属性设置为 OwnerDraw：

- Visual Basic

```
Me.C1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
```

- C#

```
this.c1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
```

3. 使用 GetCellRange 方法来设置 CellRange：

- Visual Basic

```
rng = Me.C1FlexGrid1.GetCellRange(2,  2, 4, 4)
```

- C#

```
rng = this.c1FlexGrid1.GetCellRange(2, 2, 4, 4);
```

4. 设置 **LinearGradient** 画刷的颜色和渐变的角度：

- Visual Basic

```
GradientStyleBrush = New
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.Navy, Color.Transparent, 270)
```

- C#

```
GradientStyleBrush = new
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.Navy, Color.Transparent, 270);
```

5. 添加 OwnerDrawCell 事件来在 CellRange 中画渐变：

- Visual Basic

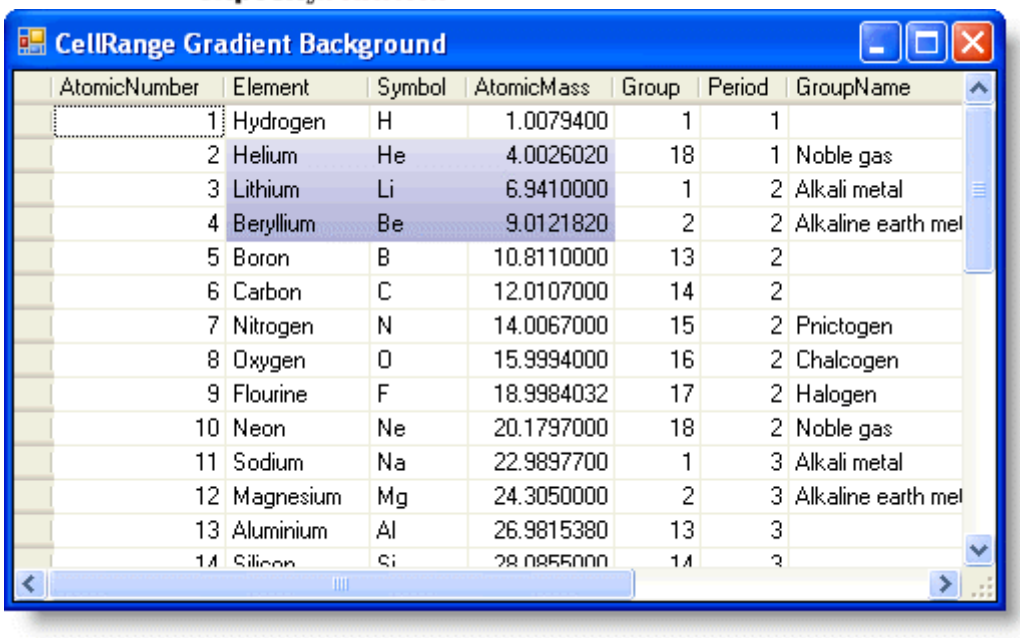
```
Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e
As C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles
C1FlexGrid1.OwnerDrawCell
' 使用渐变画刷来绘制单元格的背景色
If (e.Row >= rng.r1) And (e.Row <= rng.r2) Then
If (e.Col >= rng.c1) And (e.Col <= rng.c2) Then
' 画背景色
e.Graphics.FillRectangle(GradientStyleBrush, e.Bounds)
' 让表格绘制内容
e.DrawCell(C1.Win.C1FlexGrid.DrawCellFlags.Content)
' 完成绘制
e.Handled = True
End If
End If
End Sub
```

- C#

```
private void c1FlexGrid1_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
//使用渐变画刷来绘制单元格的背景色
if ((e.Row >= rng.r1) & (e.Row <= rng.r2))
{
if ((e.Col >= rng.c1) & (e.Col <= rng.c2))
{
//画背景色
e.Graphics.FillRectangle(GradientStyleBrush, e.Bounds);
//让表格绘制内容
e.DrawCell(C1.Win.C1FlexGrid.DrawCellFlags.Content);
//完成绘制
e.Handled = true;
}
}
}
```

本主题演示如下：

从透明到深蓝色的渐变背景色只出现在指定的单元格区域内。

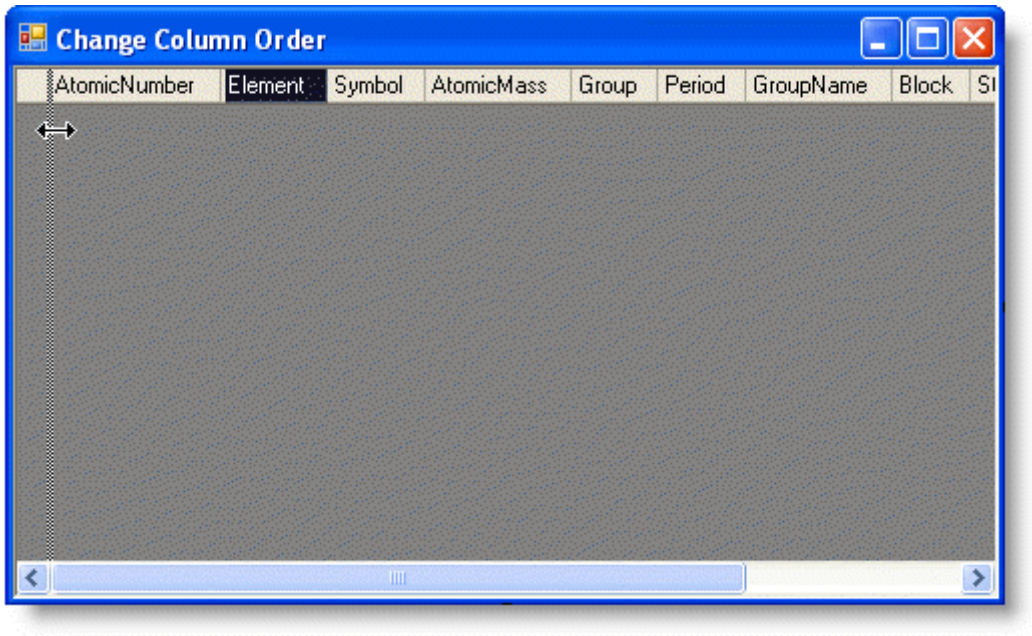

CellRange Gradient Background
AtomicNumber Element Symbol AtomicMass Group Period GroupName
1 Hydrogen H 1.0079400 1 1
2 Helium He 4.0026020 18 1 Noble gas
3 Lithium Li 6.9410000 1 2 Alkali metal
4 Beryllium Be 9.0121820 2 2 Alkaline earth mel
5 Boron B 10.8110000 13 2
6 Carbon C 12.0107000 14 2
7 Nitrogen N 14.0067000 15 2 Pnictogen
8 Oxygen O 15.9994000 16 2 Chalcogen
9 Flourine F 18.9984032 17 2 Halogen
10 Neon Ne 20.1797000 18 2 Noble gas
11 Sodium Na 22.9897700 1 3 Alkali metal
12 Magnesium Mg 24.3050000 2 3 Alkaline earth mel
13 Aluminium Al 26.9815380 13 3

7.7 在表格中改变列的顺序

要改变表格中的列顺序，可以在表格中拖动一列到一个新的位置或者使用 **C1FlexGrid 列编辑器**，还可以在代码中使用 MoveRange 方法：

在设计器中：

1. 在表格中，选择一个你想要移动的列。在本例中，Element 列将会被移动。



2. 点击并拖动这一列到左边去。一个垂直的虚线出现，这个虚线说明了这一列可以被放置在哪里。
3. 将 Element 列拖到到 AtomicNumber 列前面。

另外，可以使用表格中的 **C1FlexGrid 列编辑器**将这些列重新排序：

1. 打开 **C1FlexGrid 列编辑器**。想要了解更多关于如何访问 **C1FlexGrid 列编辑器**的细节，请查看访问 C1FlexGrid 列编辑器 章节（第 143 页）。
2. 在设计器中，选择一个你想要移动的列。在本例中，我们来移动 Element 这一列。
3. 点击并将这一列拖到左边。一个垂直的虚线出现，这个虚线说明了这一列可以被放置在哪里。
4. 将 Element 列放到 AtomicNumber 列的前面。
5. 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中，用来将第二列（在本例中是 Element 列）移动到第一列的位置上去：

- Visual Basic

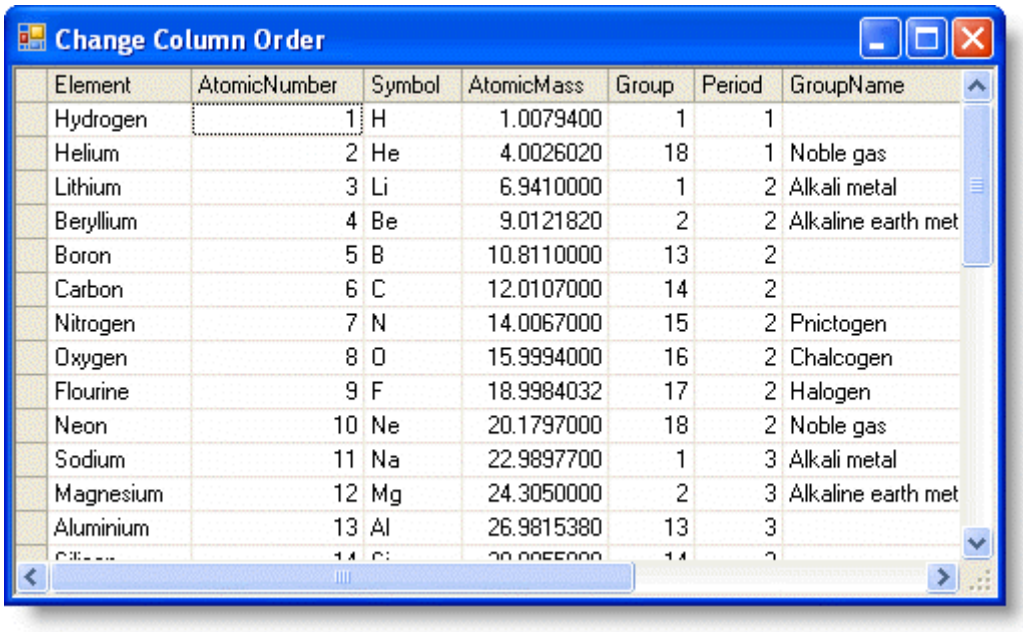
```
Me.C1FlexGrid1.Cols.MoveRange(2, 1, 1)
```

- C#

```
this.c1FlexGrid1.Cols.MoveRange(2, 1, 1);
```

本主题演示如下：

现在 Element 这一列出现在 AtomicNumber 列的前面。



Element	AtomicNumber	Symbol	AtomicMass	Group	Period	GroupName
Hydrogen	1	H	1.0079400	1	1	
Helium	2	He	4.0026020	18	1	Noble gas
Lithium	3	Li	6.9410000	1	2	Alkali metal
Beryllium	4	Be	9.0121820	2	2	Alkaline earth met
Boron	5	B	10.8110000	13	2	
Carbon	6	C	12.0107000	14	2	
Nitrogen	7	N	14.0067000	15	2	Pnictogen
Oxygen	8	O	15.9994000	16	2	Chalcogen
Flourine	9	F	18.9984032	17	2	Halogen
Neon	10	Ne	20.1797000	18	2	Noble gas
Sodium	11	Na	22.9897700	1	3	Alkali metal
Magnesium	12	Mg	24.3050000	2	3	Alkaline earth met
Aluminium	13	Al	26.9815380	13	3	

7.8 根据值过滤

想要使用 **ValueFilter**，遵循以下步骤：

1. 选中表格并点击快速标签来打开 C1FlexGrid 任务菜单。
2. 勾选 Enable Column Filtering 复选框。
3. 点击设计器连接。打开 C1FlexGrid 列编辑器。
4. 点击 AllowFiltering 属性旁边的下拉列表箭头并选择 ByValue。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

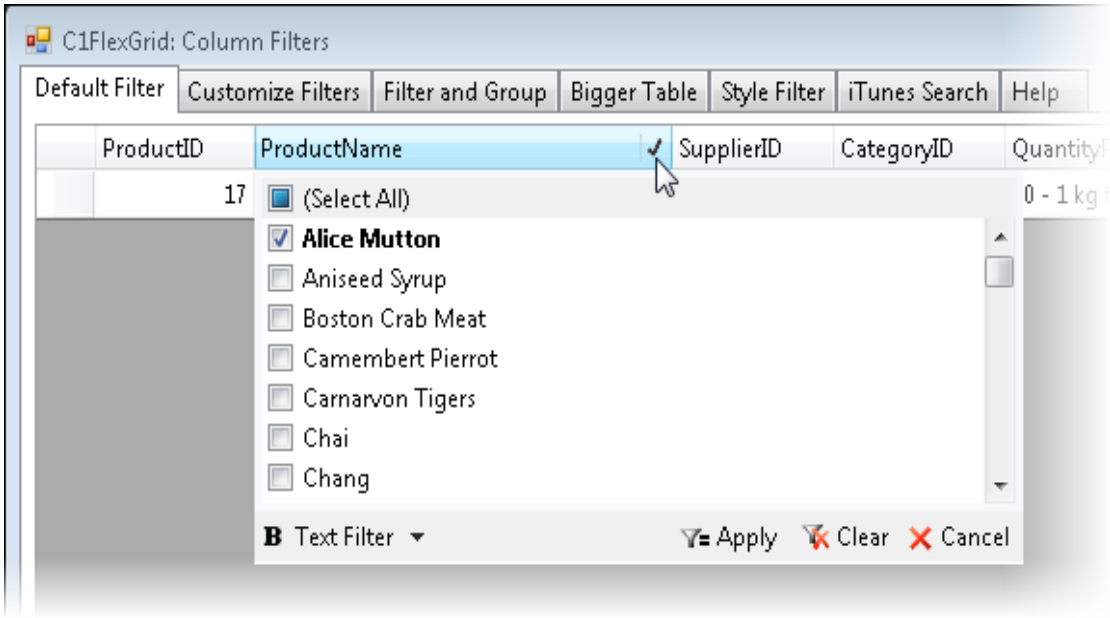
```
Me.C1FlexGrid1.AllowFiltering = True
Me.C1FlexGrid1.Cols(1).AllowFiltering = AllowFiltering.ByValue
```

- C#

```
this.c1FlexGrid1.AllowFiltering = true;
this.c1FlexGrid1.Cols[1].AllowFiltering = AllowFiltering.ByValue
```

本主题演示如下：

在本例中，第二列将会使用值排序：



7.9 根据条件过滤

想要使用**条件过滤器**，遵循以下步骤：

1. 选中表格并点击快速标签来打开 **C1FlexGrid 任务菜单**。
2. 勾选 Enable Column Filtering 复选框。
3. 点击设计器连接。打开 C1FlexGrid 列编辑器。
4. 点击 AllowFiltering 属性旁边的下拉列表箭头并选择 **ByCondition**。

在代码中：

将下面的代码添加到 **Form_Load** 事件中去：

- Visual Basic

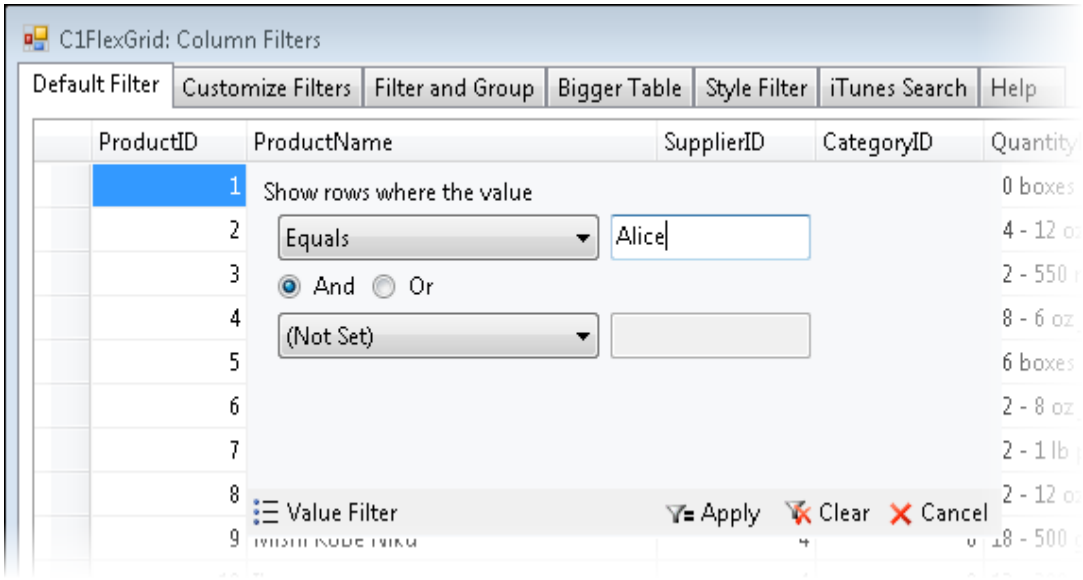
```
Me.C1FlexGrid1.AllowFiltering = True
Me.C1FlexGrid1.Cols(1).AllowFiltering = AllowFiltering.ByCondition
```

- C#

```
this.c1FlexGrid1.AllowFiltering = true;
this.c1FlexGrid1.Cols[1].AllowFiltering = AllowFiltering.ByCondition
```

本主题演示如下：

在本例中，第二列将会按照条件来排序：



7.10 更改过滤器语言

想要更改在列过滤器中使用的语言，你可以使用 Language 属性。

1. 右键点击表格并选择 **Properties** 来查看 Visual Studio 属性窗口。
2. 将 AllowFiltering 属性设置为 **True**。
3. 点击 Language 属性旁边的下拉列表箭头并选择一个语言。
4. 运行程序并点击列头上的下拉列表箭头来打开列过滤器。这一列的语言过滤将会与你在 Language 属性中指定的相同。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

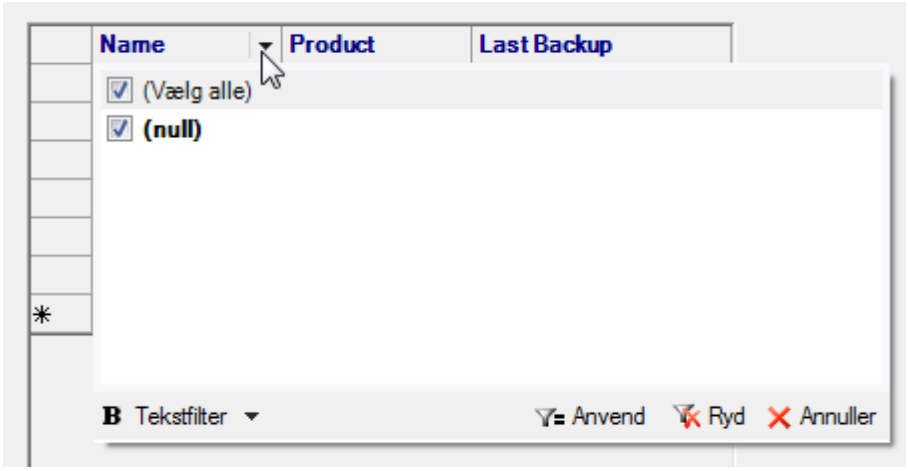
- Visual Basic

```
Me.C1FlexGrid1.AllowFiltering = True
Me.C1FlexGrid1.Language = C1.Util.Localization.Language.Danish
```

- C#

```
this.c1FlexGrid1.AllowFiltering = true;
this.c1FlexGrid1.Language = C1.Util.Localization.Language.Danish;
```

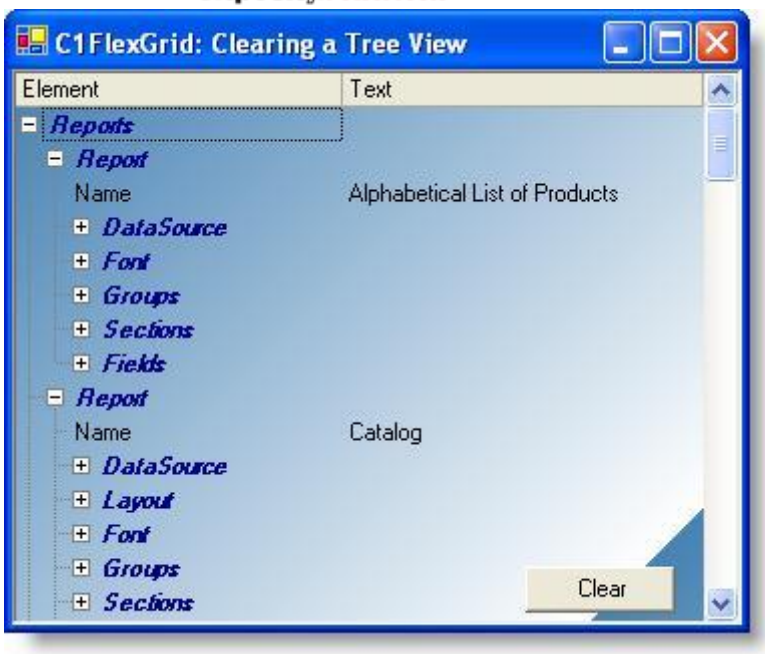
本主题演示如下：



请注意，列过滤器的语言匹配的是你 Language 属性中指定的语言。

7.11 清除树视图

想要在 C1FlexGrid 中清除树视图，将能够编辑的行数设置为 0。想要了解更多关于创建数的信息，请查看 FlexGrid for WinForms 教程和概述和汇总数据。



将下面的代码添加到 **Button1_Click** 事件中。这些代码会将可编辑的行数设置为 0，当点击 **Clear** 按钮时树视图将会被清空。

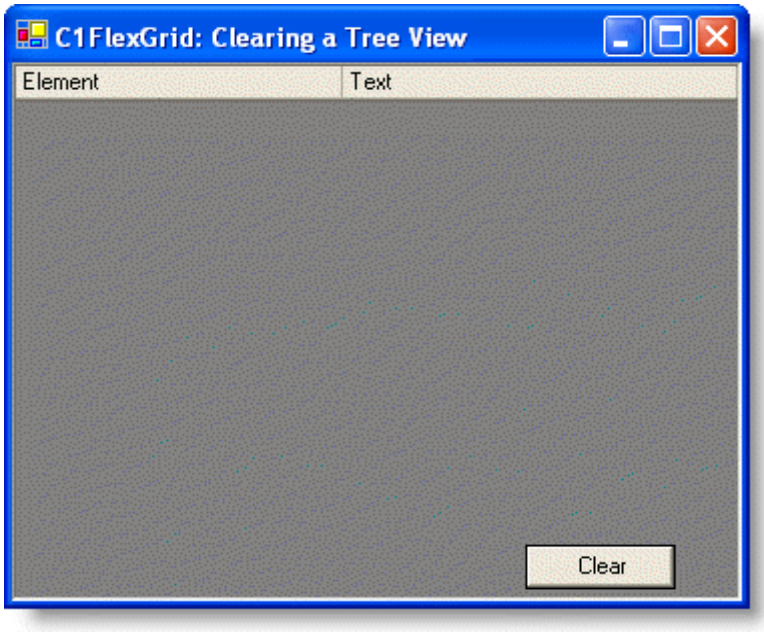
- Visual Basic

```
Me.C1FlexGrid1.Rows.Count = Me.C1FlexGrid1.Rows.Fixed
```

- C#

```
this.c1FlexGrid1.Rows.Count = this.c1FlexGrid1.Rows.Fixed;
```

本主题演示如下：



一旦你点击了这个按钮，树视图就会消失。

7.12 清除 C1FlexGrid

要清除 C1FlexGrid，请使用 Clear 方法。Clear 方法将会清除内容、样式、UserData 或所有这三者。

下面的图片展示了在没有清除内容、样式、UserData 之前的表格样子。

7.12.1 清除内容

要清除 C1FlexGrid 里的内容，添加下面的 Clear 方法。在本例中，这些代码都将被添加到 Clear Contents 按钮的 Click 事件中去。

- Visual Basic

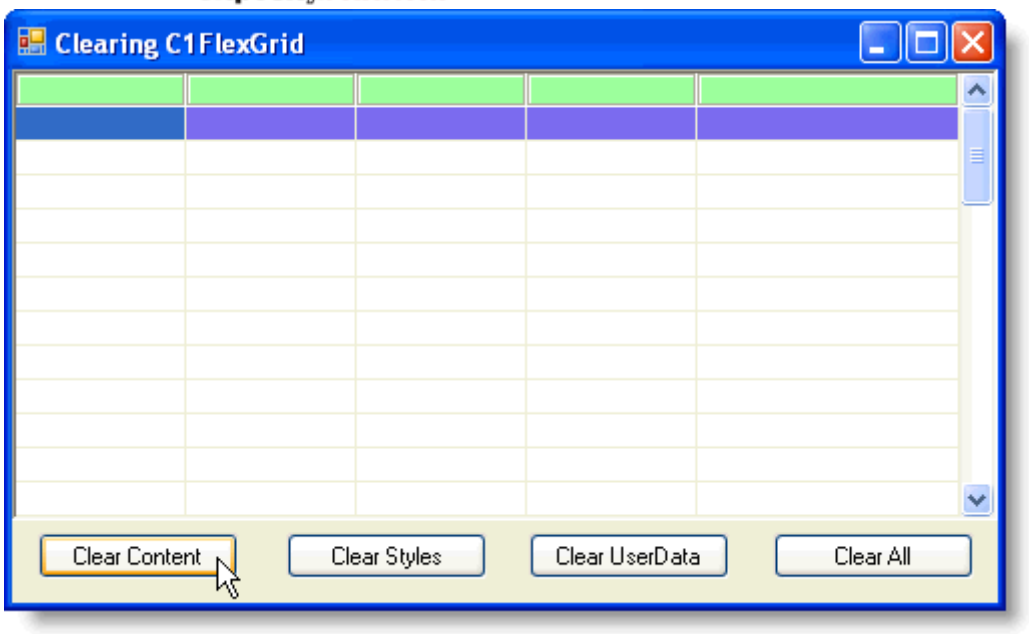
```
Me.C1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.Content)
```

- C#

```
this.c1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.Content);
```

本主题演示如下：

点击 **Clear Content** 按钮只会清除内容，不会清除表格上的样式或者 UserData。



7.12.2 清除样式

要清除 C1FlexGrid 上的显示样式和格式，添加下面的 Clear 方法。在本例中，这些代码都将被添加到 Clear Styles 按钮的 Click 事件中去。

- Visual Basic

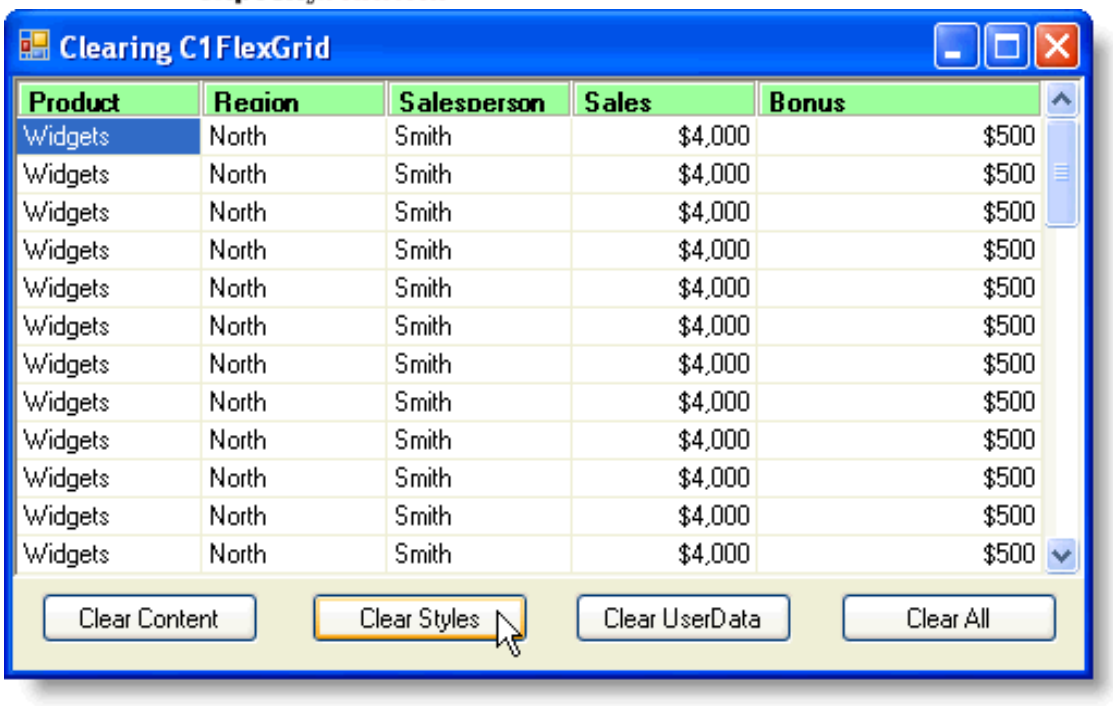
```
Me.C1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.Style)
```

- C#

```
this.c1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.Style);
```

本主题演示如下：

点击 **Clear Styles** 按钮只会清除显示样式和格式化，不会清除表格上的 UserData 和内容。



7.12.3 清除 UserData

要清除 C1FlexGrid 的 UserData，添加下面的 Clear 方法。在本例中，这些代码都将被添加到 **Clear UserData** 按钮的 **Click** 事件中去。

- Visual Basic

```
Me.C1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.UserData)
```

- C#

```
this.c1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.UserData);
```

请注意，当你点击清除 UserData 时，画面上将不会出现任何变化，除非 UserData 存储了这个程序很有用的内容。

7.12.4 清除内容、样式和 UserData

想要一次性清除 C1FlexGrid 上的内容、样式和 UserData，添加下面的 **Clear** 方法。在本例中，这些代码都将被添加到 **Clear All** 按钮的 Click 事件中去。

- Visual Basic

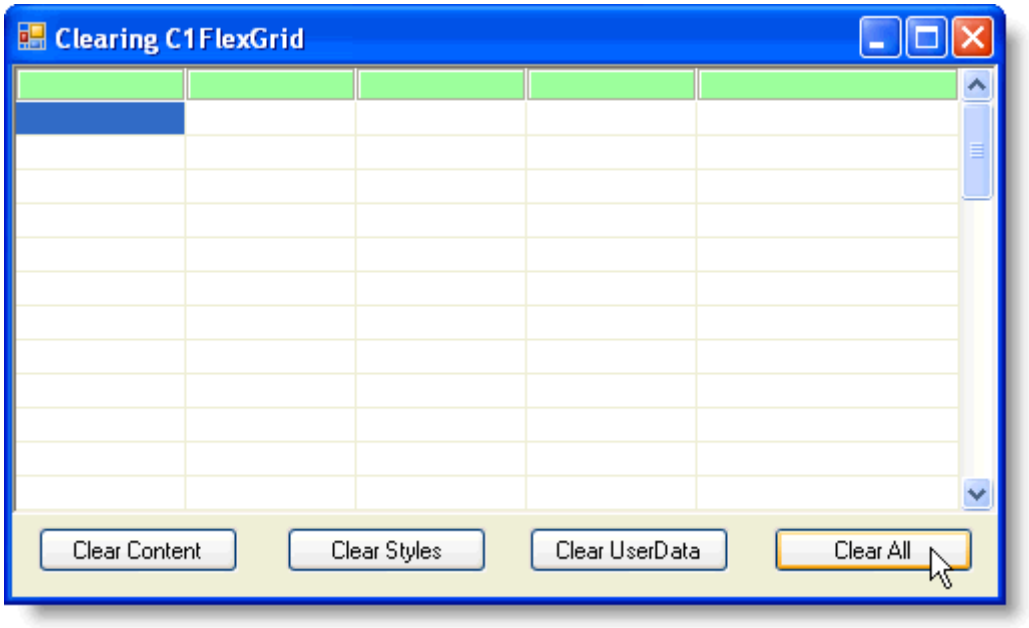
```
Me.C1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.All)
```

- C#

本主题演示如下：

```
this.c1FlexGrid1.Clear(C1.Win.C1FlexGrid.ClearFlags.All);
```

点击 Clear All 将会清除表格上的内容、样式和 UserData，只留下一个空的表格。



7.13 将列上的字母转换为大写字母

要将列的字母从小写转换为大写，添加下面的 SetupEditor 事件到窗体上：

- Visual Basic

```
Private Sub C1FlexGrid1_SetupEditor(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles C1FlexGrid1.SetupEditor
If Me.C1FlexGrid1.Cols(e.Col).Name = "UCASECOL" Then
Dim tb As TextBox = Me.C1FlexGrid1.Editor
tb.CharacterCasing = CharacterCasing.Upper
End If
End Sub
```

- C#

7.14 使用视觉样式来自定义外观

想要使用视觉样式自定义 C1FlexGrid 的外观 ，将 VisualStyle 属性设置为 Custom、Office2007Black、 Office2007Blue、 Office2007Silver、Office2010Blue、Office2010Black、Office2010Silver、或者 System。这个属性可以在设计器中或在代码中设置。下面的列表描述了每种视觉样式：

视觉样式	描述

```
private void c1FlexGrid1_SetupEditor(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
if (this.c1FlexGrid1.Cols[e.Col].Name == "UCASECOL")
{
TextBox tb = this.c1FlexGrid1.Editor as TextBox;
tb.CharacterCasing = CharacterCasing.Upper;
}
}
```

Custom	不适用任何视觉样式，只使用样式和属性来构造控件。
Office2007Black	不适用任何视觉样式，只使用样式和属性来构造控件。
Office2007Blue	不适用任何视觉样式，只使用样式和属性来构造控件。
Office2007Silver	使用基于 Office 2007 的银色配色方案来构造控件的外观。
System	使用基于当前系统配色方案来构造控件的外观。
Office2010Black	使用基于当前系统 的银色配色方案来构造控件的外观。
Office2010Silver	使用基于 Office 2010 的银色配色方案来构造控件的外观。
Office2010Blue	使用基于 Office 2010 的蓝色配色方案来构造控件的外观。

在设计器中：

在属性窗口中找到 VisualStyle 属性，将它设置为 Custom、Office2007Black、Office2007Blue、Office2007Silver、Office2010Blue、Office2010Black、Office2010Silver 或者 System。在本例中，VisualStyle 属性将被设为 Office2007Blue。

在代码中：

将代码添加到 Form_Load 事件中，将 VisualStyle 属性设置为 Custom、Office2007Black、Office2007Blue、Office2007Silver、Office2010Blue、Office2010Black、Office2010Silver 或者 System。下面的代码将 VisualStyle 属性设

置为 Office2007Blue：

- Visual Basic

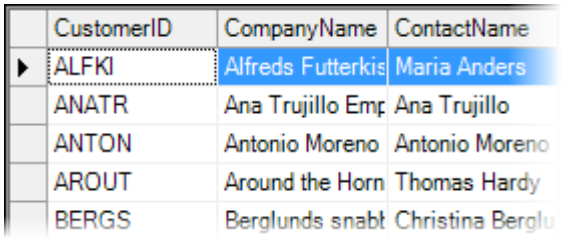
```
Me.C1FlexGrid1.VisualStyle = C1.Win.C1FlexGrid.VisualStyle.Office2007Blue
```

- C#

```
this.c1FlexGrid1.VisualStyle =
C1.Win.C1FlexGrid.VisualStyle.Office2007Blue;
```

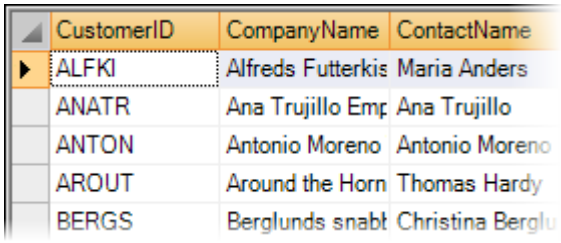
自定义视觉样式

没有视觉样式应用的样子：



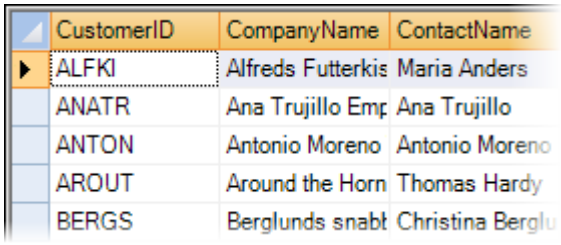
Office2007Black 视觉样式

Office 2007 黑色方案：



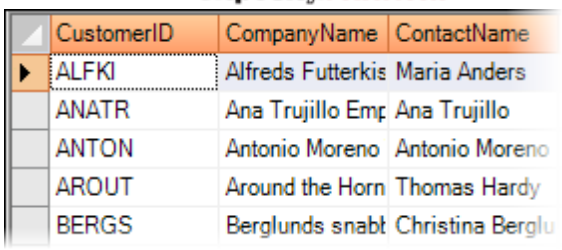
Office2007Blue 视觉样式

Office 2007 蓝色方案：



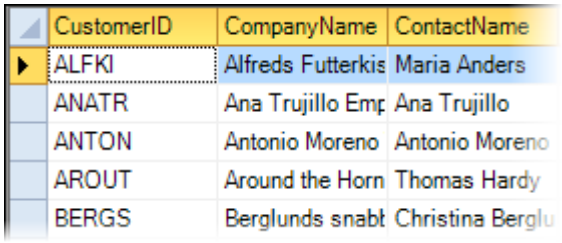
Office2007Silver 视觉样式

Office 2007 银色方案：



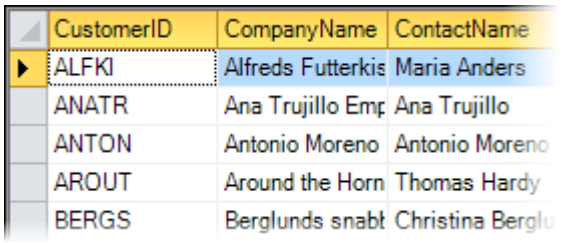
Office2010Blue 视觉样式

Office 2010 蓝色方案：



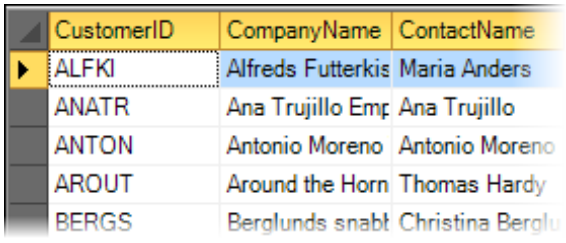
Office2010Silver 视觉样式

Office 2010 银色方案：



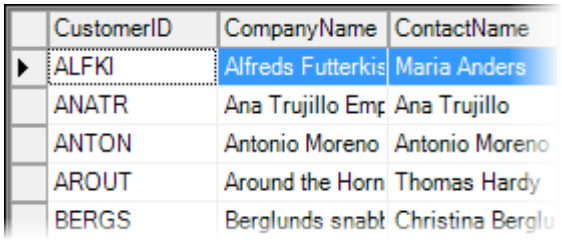
Office2010Black 视觉样式

Office 2010 黑色方案：



System 视觉样式

当前系统设置：



7.15 只在单元格中输入数字

想要只允许在单元格中输入数字，请使用 KeyPressEdit 事件并当按下无效按键时，将它的 e.Handled 参数设置为 **True**。请使用下面的代码将第四列设置为只能按数字、BACKSPACE、DELETE 和·键：

- Visual Basic

```
    Private Sub C1FlexGrid1_KeyPressEdit(ByVal sender As Object, ByVal e As
    C1.Win.C1FlexGrid.KeyPressEditEventArgs) Handles
C1FlexGrid1.KeyPressEdit
    If e.Col = 3 Then
    ' 如果输入不在 0-9、 PERIOD、 DELETE 或 BACKSPACE 之间。
    If Not (e.KeyChar = Chr(48) Or e.KeyChar = Chr(49) Or _
    e.KeyChar = Chr(50) Or e.KeyChar = Chr(51) Or _
    e.KeyChar = Chr(52) Or e.KeyChar = Chr(53) Or _
    e.KeyChar = Chr(54) Or e.KeyChar = Chr(55) Or _
    e.KeyChar = Chr(56) Or e.KeyChar = Chr(57) Or _
    e.KeyChar = Chr(46) Or e.KeyChar = Chr(127) Or _
    e.KeyChar = Chr(8)) Then
    ' 禁止输入非法的按键值到控件上
    e.Handled = True
    End If
    End If
    End Sub
```

- C#

```
    private void c1FlexGrid1_KeyPressEdit(object sender,
    C1.Win.C1FlexGrid.KeyPressEditEventArgs e)
    {
    if( e.Col == 3 )
    {
    //如果输入不在 0-9、 PERIOD、 DELETE 或 BACKSPACE 之间
    if( !((e.KeyChar == 48) || (e.KeyChar == 49) ||
    (e.KeyChar == 50) || (e.KeyChar == 51) || (e.KeyChar == 52) ||
    (e.KeyChar == 53) || (e.KeyChar == 54) || (e.KeyChar == 55) ||
    (e.KeyChar == 56) || (e.KeyChar == 57) || (e.KeyChar == 46) ||
    (e.KeyChar == 127) || (e.KeyChar == 8)))
    //禁止输入非法的按键值到控件上
    e.Handled = true;
    }
    }
```

7.16 格式化单元格

通过格式化单元格操作，允许您控制单个单元格或多单元格的数据输入和表现形式。下面的话题向你展示了如何设置单元格为只读，如何将单元格格式化为显示货币值，如何将单元格格式化为根据内容来显示相应的格式。

7.16.1 将单元格设置为只读

要想将单个或多个单元格设置为只读，请使用 BeforeEdit 事件。

将单个单元格设置为只读

你可以将表格中任何一个单元格设置为只读，以便其中的数据不能被更改。例如，添加下面的代码，将第一行第一列的单元格设置为只读：

- Visual Basic

```
Private Sub C1FlexGrid1_BeforeEdit(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles C1FlexGrid1.BeforeEdit
If e.Row = 1 And e.Col = 1 Then
e.Cancel = True
End If
End Sub
```

- C#

```
private void c1FlexGrid1_BeforeEdit(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
if (e.Row == 1 & e.Col == 1)
{
e.Cancel = true;
}
}
```

将多单元格设置为只读

你也许想要将表格中多个单元格一次性设置为只读。假设你有一个可以编辑的表格，用于输入客户的信息，如客户 ID、地址、订单。其中订单和地址会改变，但客户 ID 不会。下面的代码假定你有一个九行的表格，你想将第一列（客户 ID 列）的所有行，设置为只读的。请输入以下代码：

- Visual Basic

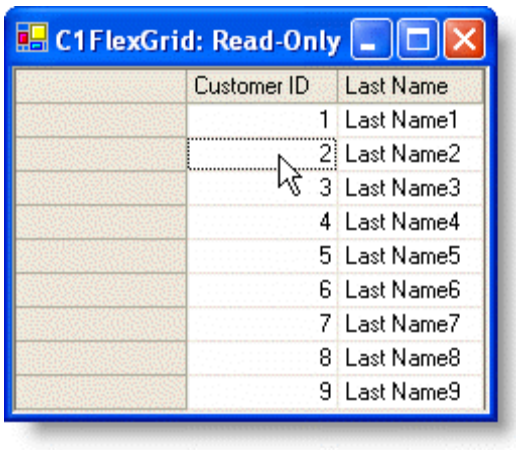
```
Private Sub C1FlexGrid1_BeforeEdit(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles C1FlexGrid1.BeforeEdit
Dim i As Integer
For i = 1 To 9
If e.Col = 1 And e.Row = i Then
e.Cancel = True
End If
Next
End Sub
```

- C#

```
private void c1FlexGrid1_BeforeEdit(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
for (int i = 1; i <= 9; i++)
{
if (e.Col == 1 & e.Row == i)
{
e.Cancel = true;
}
}
}
```

本主题演示如下：

请注意，在 CustomerID 列中，没有一行是可以编辑的。



7.16.2 将单元格格式化为显示十进制内容

想要将单元格格式化为只显示十进制数字，请在设计器中或者在代码中设置 Format 属性。在本例中，已经在第一列中输入的数字将被格式化为十进制的金额。

在设计器中：

1. 在表格中选择 Column 1。这将会打开 Column 1 的**列任务**菜单。
2. 点击 Format String 旁边的省略号按钮来打开 Format String 对话框。
3. 在 Format type 下面选择 Custom 之后将它设置为 Custom format：$#,##0.00。
4. 点击 OK 来关闭 **Format String** 对话框。

另外， Format 属性也可以使用 C1FlexGrid 列编辑器来设置：

1. 打开 **C1FlexGrid 列编辑器**。想要了解更多关于如何访问 **C1FlexGrid 列编辑器**的信息，请查看访问 C1FlexGrid 列编辑器 章节（第 143 页）。
2. 在右侧窗格中选择 Column 1 并在左侧窗格中将 **Format** 属性设置为**$#,##0.00** 。
3. 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
Me.C1FlexGrid1.Cols(1).Format = "$#,##0.00"
```

- C#

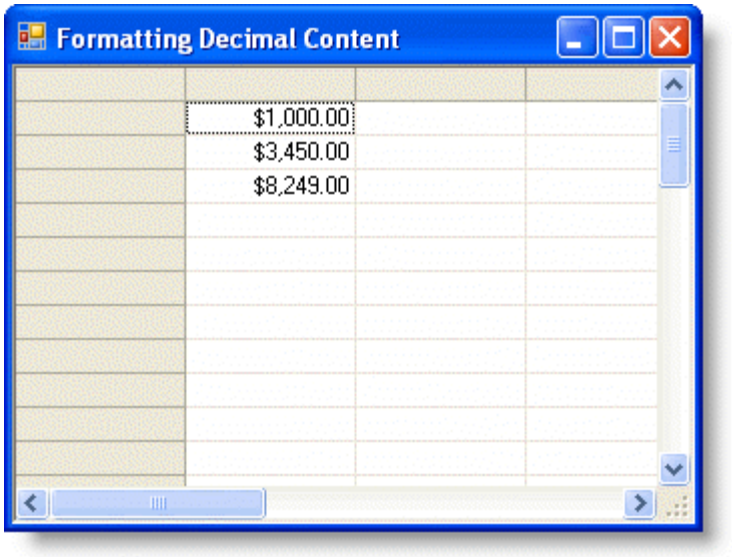
```
this.c1FlexGrid1.Cols[1].Format = "$#,##0.00";
```

注意：格式串中的**说明符**完全遵守标准的.NET 约定。使用',' 表示千位分隔

符，使用'.'表示十进制数字，跟区域无关。

本主题演示如下：

在本例中，在第一列中的数字将会转换为美元金额。



7.16.3 基于内容格式化单元格

要有条件的格式化基于内容的单元格，请创建一个新的样式并使用 CellChanged 事件。

1. 创建一个新的 **CellStyle** 名为 LargeValue 并将它的 **BackColor** 属性设置为 **Gold**。

在设计器中：

打开 C1FlexGrid 样式编辑器。想要了解更多关于如何访问 C1FlexGrid 样式编辑器的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。

1. 点击 Add 创建一个新样式。
2. 双击 CustomStyle1，将它重命名为 LargeValue，完成之后请按下 ENTER。
3. 在右侧窗格中选择 BackColor 属性并将它设置为 Gold。
4. 找到 Font 属性之后点击省略号按钮来打开 Font 对话框。
5. 将 Font style 设置为 Italic。
6. 点击 OK 来关闭 Font 对话框。
7. 点击 OK 来关闭 C1FlexGrid 样式编辑器。

在代码中：

1. 将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
' 为大数据创建一个自定义样式。
Dim cs As C1.Win.C1FlexGrid.CellStyle
cs = Me.C1FlexGrid1.Styles.Add("LargeValue")
cs.Font = New Font(Font, FontStyle.Italic)
cs.BackColor = Color.Gold
```

- C#

```
//为大数据创建一个自定义样式。
C1.Win.C1FlexGrid.CellStyle cs;
cs = this.c1FlexGrid1.Styles.Add("LargeValue");
cs.Font = new Font(Font, FontStyle.Italic);
cs.BackColor = Color.Gold;
```

2. 在 CellChanged 事件中基于单元格内容进行格式化

- Visual Basic

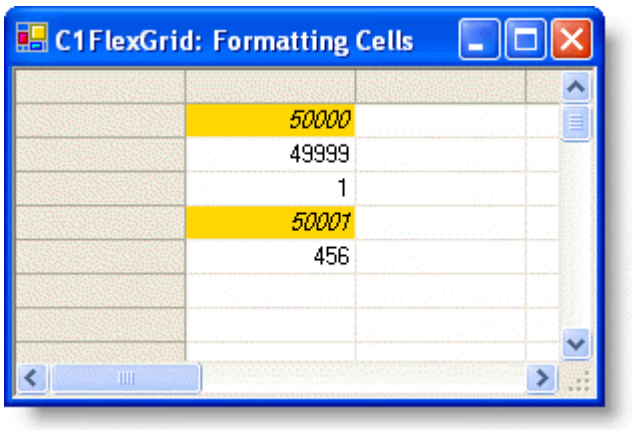
```
' 根据单元格的内容来格式化。
Private Sub C1FlexGrid1_CellChanged(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles C1FlexGrid1.CellChanged
' 标记当货币值> 50,000 作为 LargeValues。
' 将其他的样式重置为 Nothing。
Dim cs As C1.Win.C1FlexGridCellStyle
If Val(Me.C1FlexGrid1(e.Row, e.Col)) >= 50000 Then
cs = Me.C1FlexGrid1.Styles("LargeValue")
End If
Me.C1FlexGrid1.SetCellStyle(e.Row, e.Col, cs)
End Sub
```

- C#

```
//根据单元格的内容来格式化。
private void c1FlexGrid1_CellChanged(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
//标记当货币值> 50,000 作为 LargeValues。
//其他的样式重置为 Nothing。
C1.Win.C1FlexGrid.CellStyle cs;
if (Val(this.c1FlexGrid1(e.Row, e.Col)) >= 50000)
{
cs = this.c1FlexGrid1.Styles("LargeValue");
}
this.c1FlexGrid1.SetCellStyle(e.Row, e.Col, cs);
}
```

本主题演示如下：

在本例中，单元格中如果包含大于等于 50,000 时将会高亮斜体为金色。



7.17 格式化边框样式

格式化边框样式允许您自定义表格的外观。无论是控件还是表格的边框的样式都可以被设置。

7.17.1 格式化控件的边框样式

想要格式化控件的边框样式，请将 ScrollableControl.BorderStyle 属性设置为 **Fixed3D、FixedSingle、Light3D、None** 或者 **XpThemes**。这个属性在设计器中或者在代码中设置。下面的列表描述了每一种边框样式：

Border	描述
Fixed3D	一个三维的边框。这是默认值。
FixedSingle	单线条的边框。
Light3D	一个轻微凹陷的边框。
None	没有边框。
XpThemes	一个使用 XP 样式的边框。

在设计器中：

在属性窗口中找到 BorderStyle 属性并将它设置为 Fixed3D、FixedSingle、Light3D、None 或者 XpThemes。在本例中，BorderStyle 属性将被设置为 Fixed3D。

在代码中：

将代码添加到 Form_Load 事件中，设置 BorderStyle 属性为 Fixed3D、FixedSingle、Light3D、None 或者 XpThemes。下面的代码将 BorderStyle 属性设置为 Fixed3D：

- Visual Basic

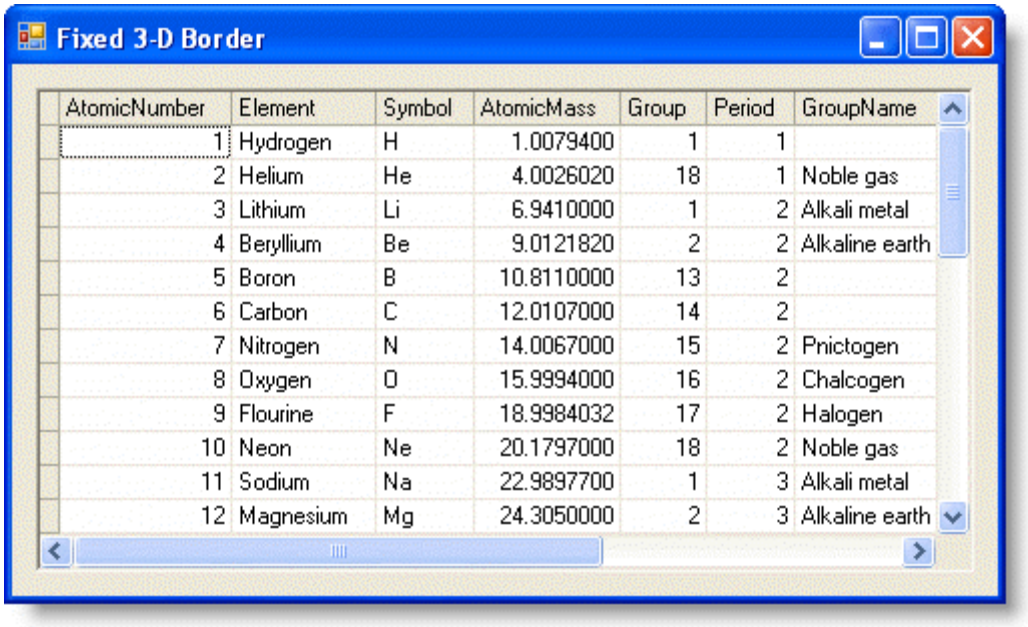
```
Me.C1FlexGrid1.BorderStyle =
C1.Win.C1FlexGrid.Util.BaseControls.BorderStyleEnum.Fixed3D
```

- C#

```
this.c1FlexGrid1.BorderStyle =
C1.Win.C1FlexGrid.Util.BaseControls.BorderStyleEnum.Fixed3D;
```

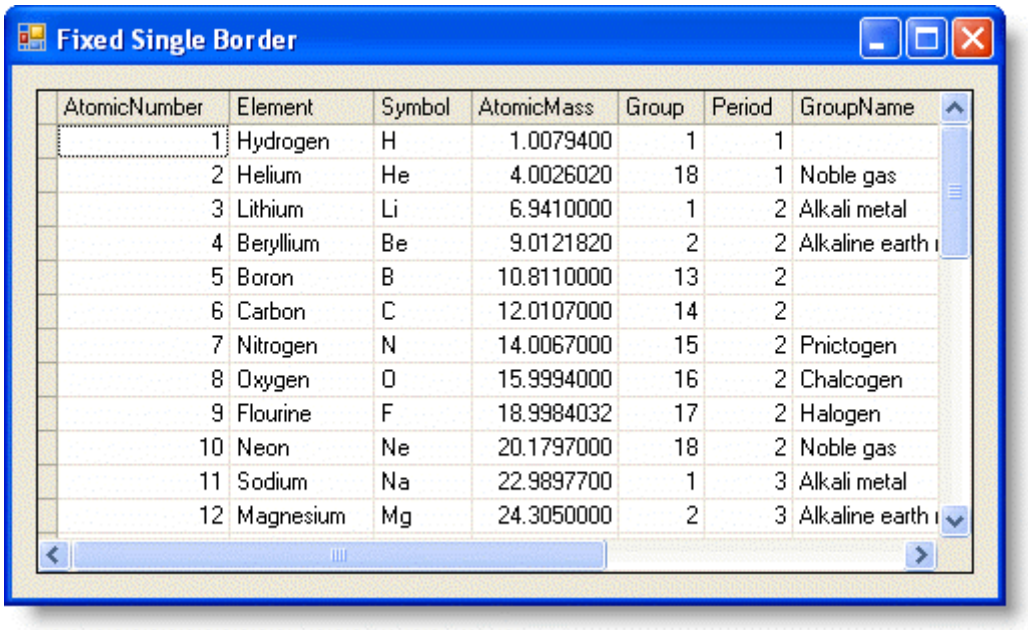
三维边框

边框将是 3D 立体的。



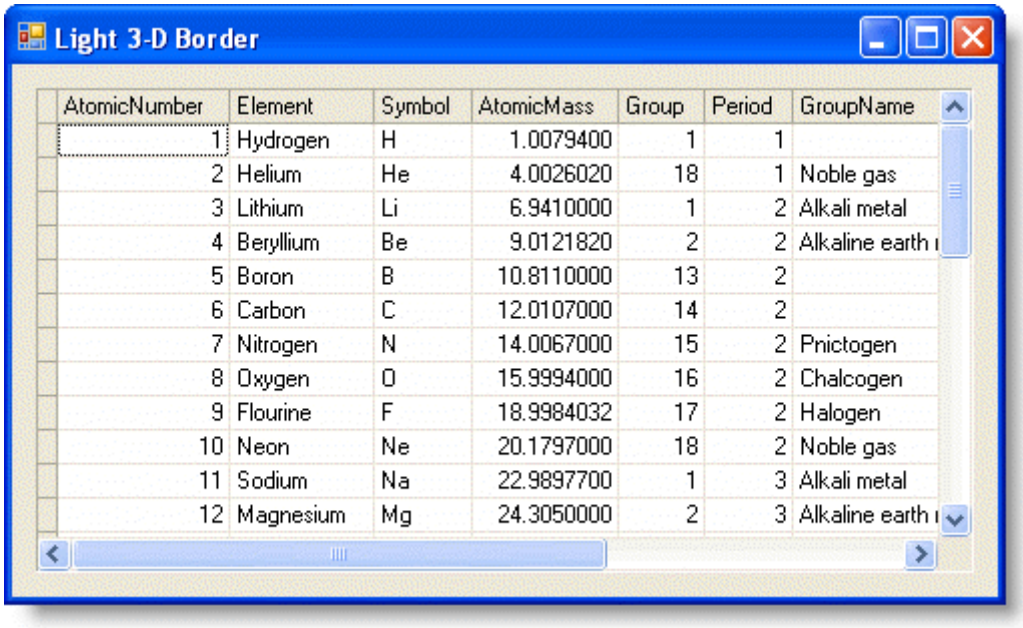
单线条边框

边框将是单线条的。



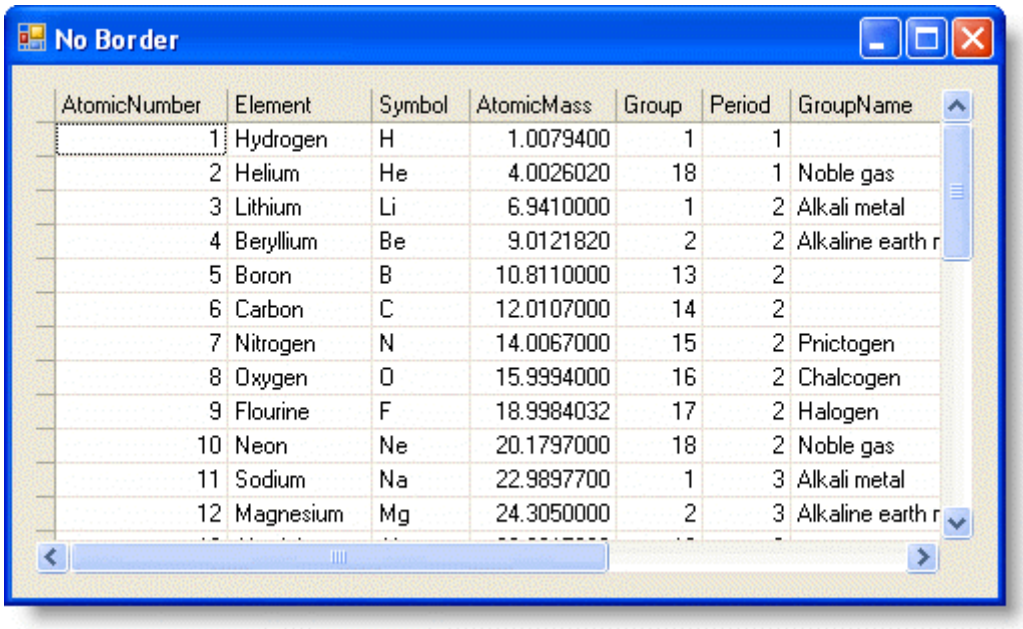
轻微凹陷的边框

边框将是轻微凹陷并且是三维的。



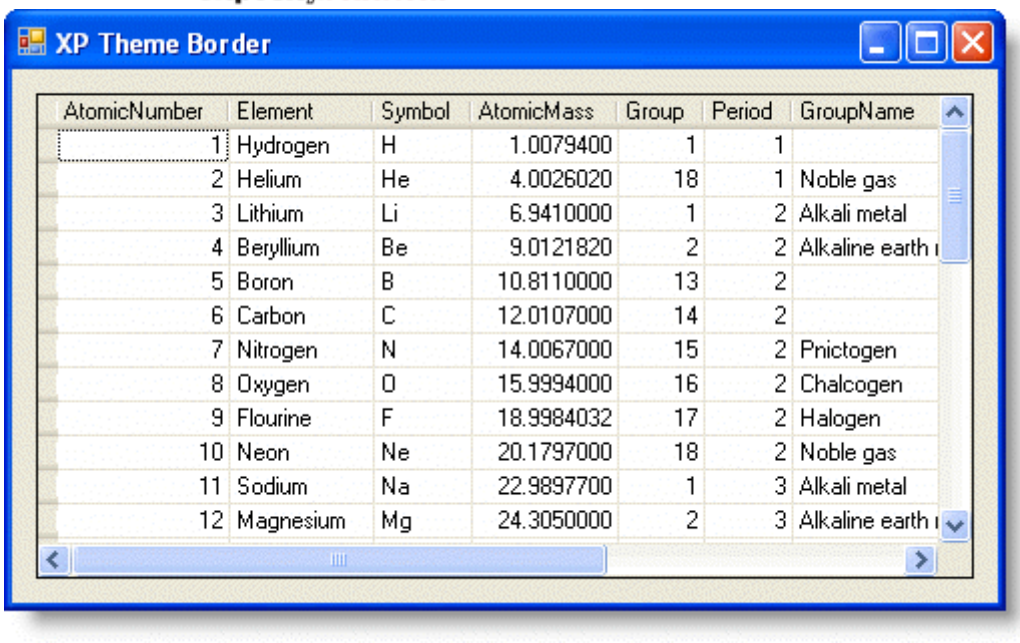
无边框

将没有任何边框。



XP Theme 边框

边框将是 XP 样式的。



边框	描述
Dotted	虚线边框。
Double	双层边框。
Fillet	圆角边框。
Flat	实心平边框。
Groove	凹陷型边框。
Inset	分段边框。
None	没有边框。这是默认设置。
Raised	突起边框。

7.17.2 格式化显示表格的边框样式

想要格式化表格的边框样式，请将 Style 设置为 **Dotted、 Double、 Fillet、 Flat、 Groove、 Inset、 None** 或者 **Raised**。这个属性可以使用 C1FlexGrid 的内建样式，它可以在设计器中或者在代码中被设置。下面的列表描述了每一种边框样式。

在设计器中：

1. 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 C1FlexGrid 样式编辑器的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
2. 在内置样式中选择 Normal 。在右边的列中，找到 Border 属性并展开它。设置 Style 属性为 Dotted、 Double、 Fillet、 Flat、 Groove、 Inset、 None 或者 Raised。在本例中，Style 属性被设置为 Dotted。
3. 点击 OK 来关闭编辑器

在代码中：

将下面的代码添加到 **Form_Load** 事件中，用来将 **Style** 属性指定为 **Normal**。下面的代码将 **Style** 设置为 **Dotted**。

- Visual Basic

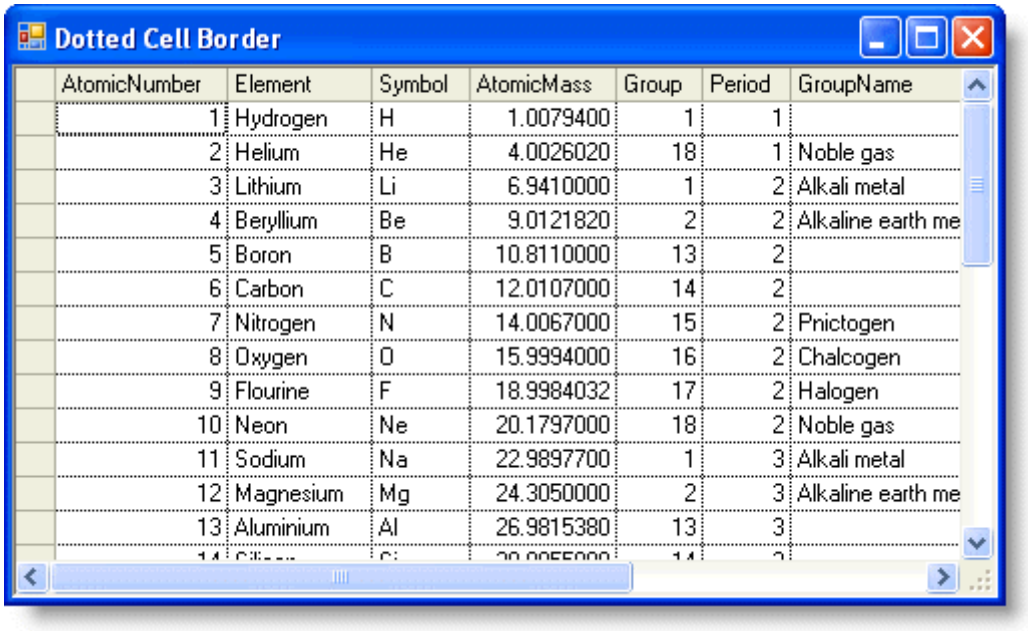
```
Me.C1FlexGrid1.Styles("Normal").Border.Style =
C1.Win.C1FlexGrid.BorderStyleEnum.Dotted
```

- C#

```
this.c1FlexGrid1.Styles["Normal"].Border.Style =
C1.Win.C1FlexGrid.BorderStyleEnum.Dotted;
```

Dotted 边框

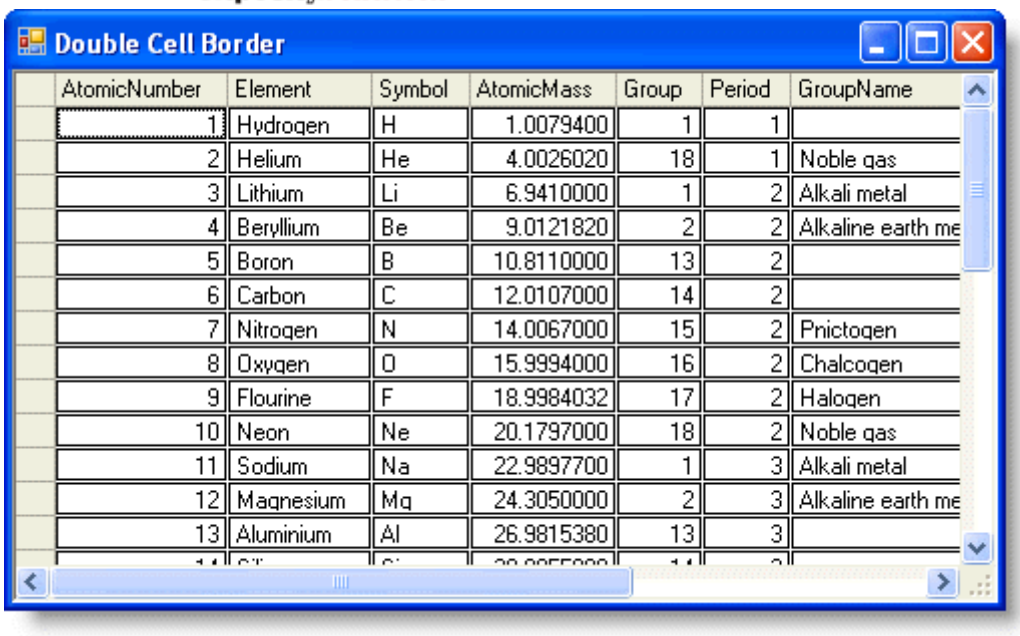
单元格的边框将会是虚线。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth me
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth me
13	Aluminium	Al	26.9815380	13	3	

Double 边框

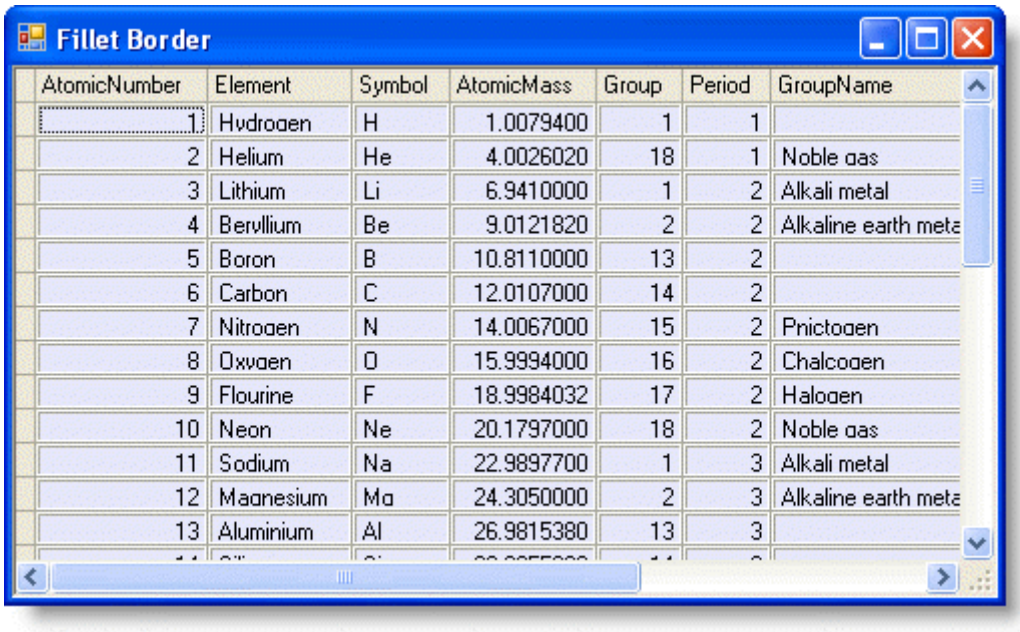
单元格边框将会是双线。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth me
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth me
13	Aluminium	Al	26.9815380	13	3	

Fillet 边框

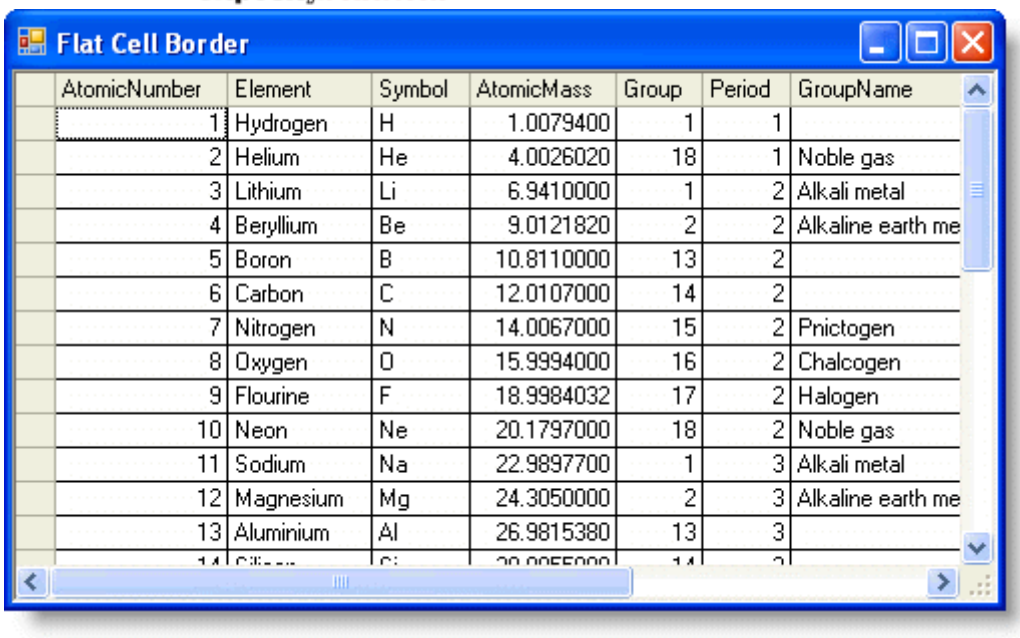
单元格边框是圆角的。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth meta
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth meta
13	Aluminium	Al	26.9815380	13	3	

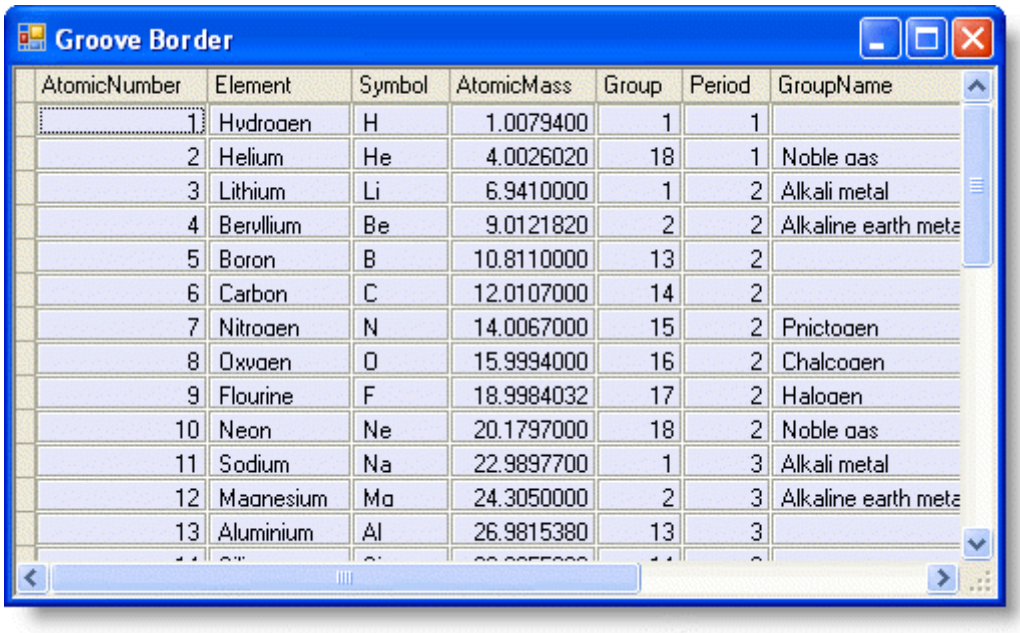
Flat 边框

单元格边框是一条平铺的线。



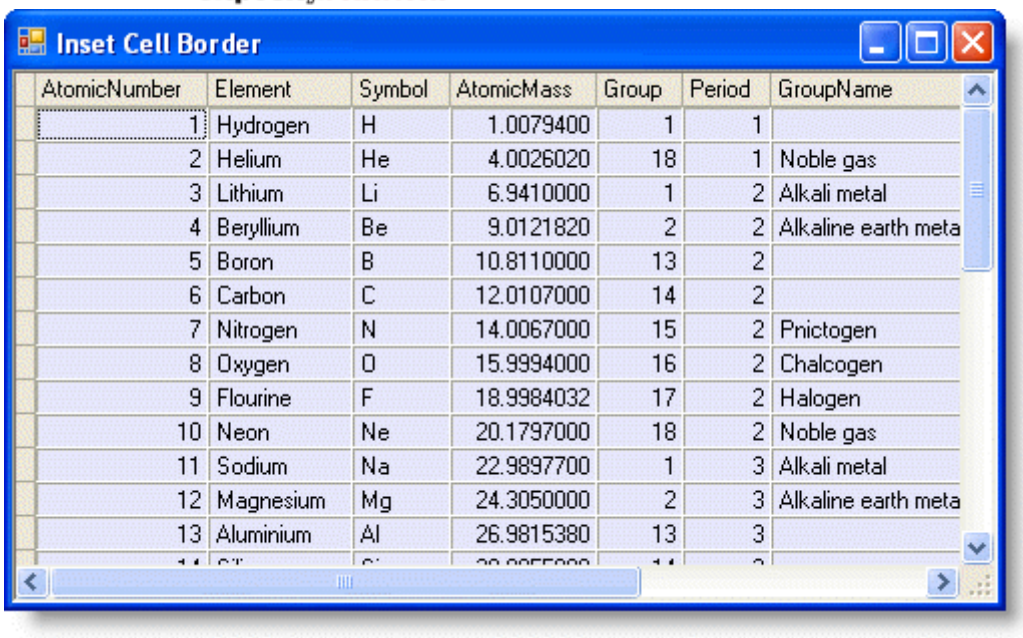
Groove 边框

单元格边框将是凹陷的。



Inset 边框

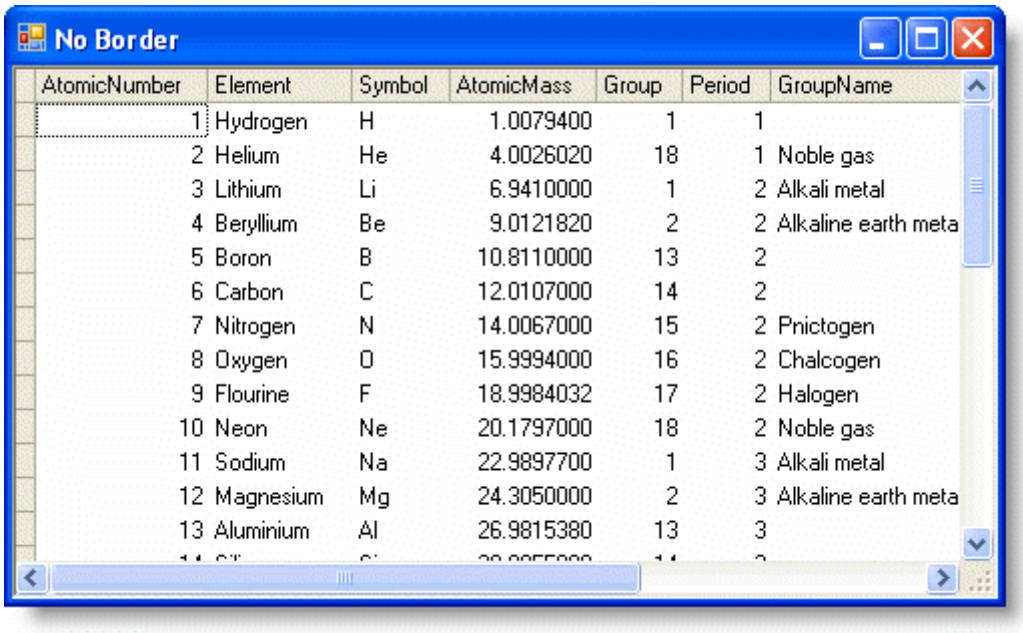
单元格边框将是嵌入式的。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth meta
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth meta
13	Aluminium	Al	26.9815380	13	3	

无边框

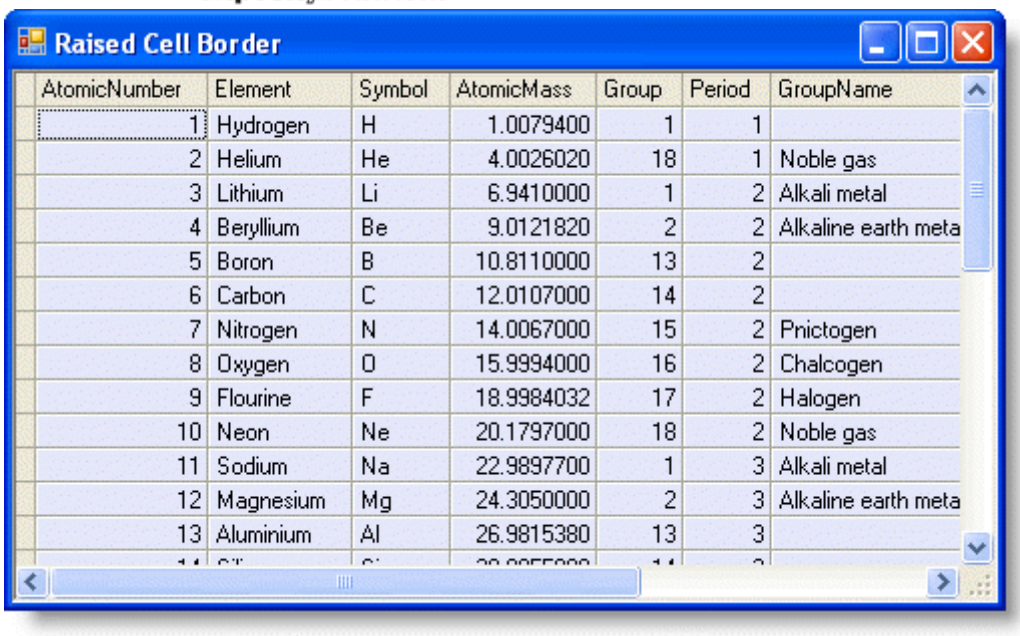
将没有边框。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth meta
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth meta
13	Aluminium	Al	26.9815380	13	3	

Raised 边框

单元格边框将是突起的。


Raised Cell Border
AtomicNumber Element Symbol AtomicMass Group Period GroupName
1 Hydrogen H 1.0079400 1 1
2 Helium He 4.0026020 18 1 Noble gas
3 Lithium Li 6.9410000 1 2 Alkali metal
4 Beryllium Be 9.0121820 2 2 Alkaline earth meta
5 Boron B 10.8110000 13 2
6 Carbon C 12.0107000 14 2
7 Nitrogen N 14.0067000 15 2 Pnictogen
8 Oxygen O 15.9994000 16 2 Chalcogen
9 Flourine F 18.9984032 17 2 Halogen
10 Neon Ne 20.1797000 18 2 Noble gas
11 Sodium Na 22.9897700 1 3 Alkali metal
12 Magnesium Mg 24.3050000 2 3 Alkaline earth meta
13 Aluminium Al 26.9815380 13 3

7.18 冻结行和列

想要允许用户通过鼠标来冻结行和列，设置 AllowFreezing 属性到 **Columns** 上将会只冻结列，设置到 **Rows** 上将会只冻结行，或者 **Both** 来冻结两者。反过来，想要去掉冻结，请将 AllowFreezing 设置为 **None**，这当然也是默认值。这个属性可以在设计器中或者在代码中来设置。

在设计器中：

在属性窗口中，找到 **AllowFreezing** 属性并将它设置为 **Both**。

在代码中：

将下面的代码添加到 Form_Load 事件中，将 **AllowFreezing** 设置为 **Both**：

- Visual Basic

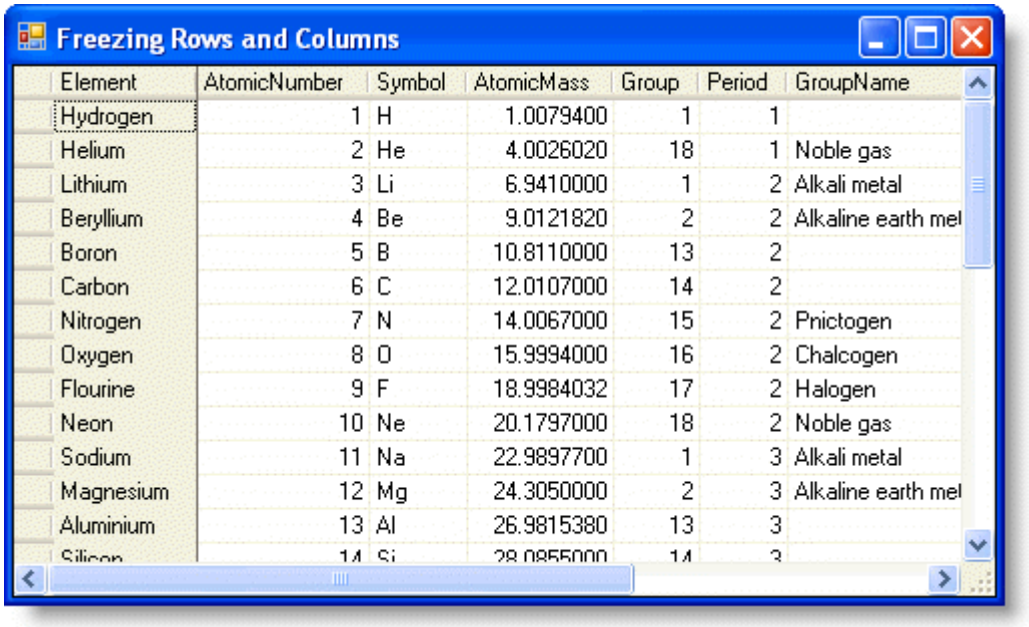
```
    Me.C1FlexGrid1.AllowFreezing =
C1.Win.C1FlexGrid.AllowFreezingEnum.Both
```

- C#

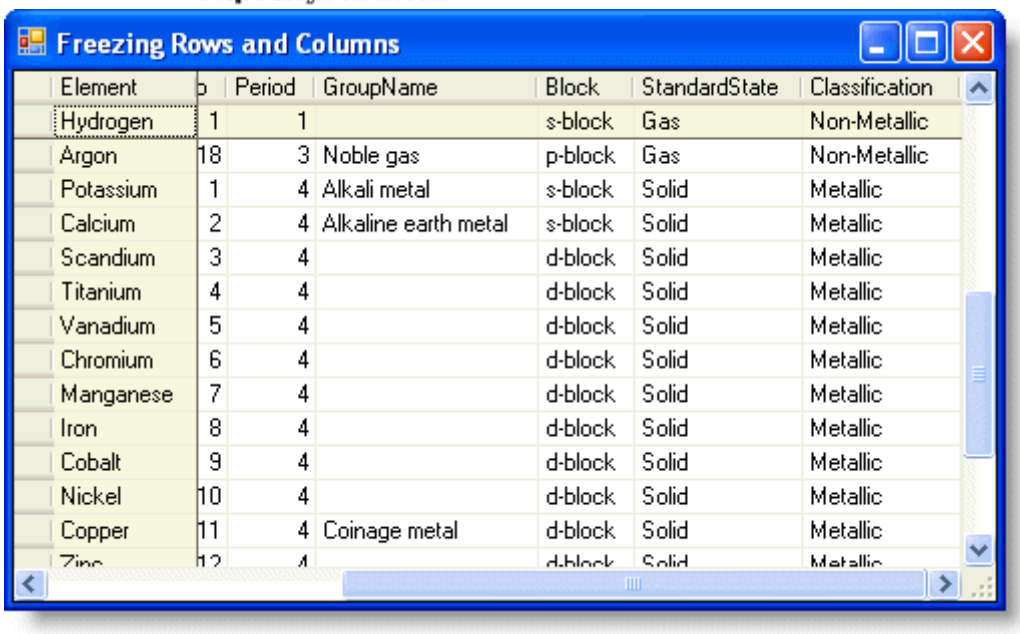
```
    this.c1FlexGrid1.AllowFreezing =
C1.Win.C1FlexGrid.AllowFreezingEnum.Both;
```

本主题演示如下：

当光标变为行锁状态图标 或者列锁状态图标 ，点击并拖动鼠标到想要冻结的列或行上。在本例中，Element 列将会被冻结，它将会一直保持在界面上，无论是否滚动到最右边。



在本例中，包含了 Hydrogen 的这一行将会被冻结，它会一直保持在界面上，无论是否滚动到最底。



注意： 将 **AllowFreezing** 属性设置为 **Both** 将会允许同时冻结列和行，例如上图。

7.19 读取和保存 Open XML 文件

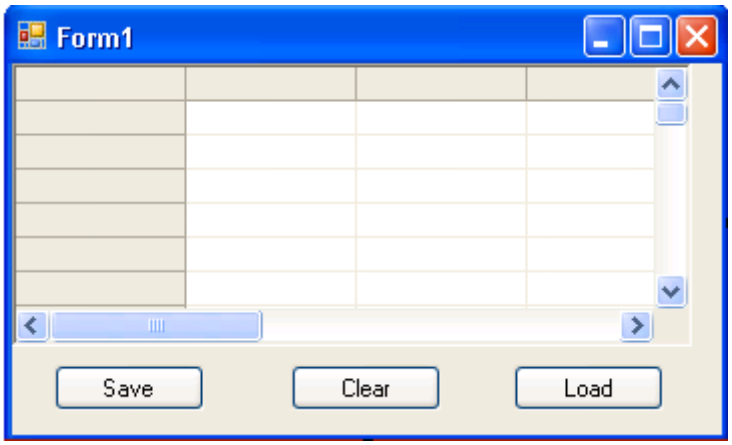
你可以使用 C1FlexGrid 控件来读取和保存 Microsoft Excel 2007 Open XML 文件。Open XML 是由微软推出 Office2007 时带来的一个开放的，基于标准的格式。Open XML 文件包含一些使用 Zip 压缩的 XML 文件。因为这些文件被压缩过，所以 Open XML 比通常的文件体积要小（例如 XLS 文件）。

默认情况下，**C1FlexGrid** 中的 **Load** 和 **Save** 方法会根据文件的扩展名选择适当的文件格式，任何的以“XLSX”或者“ZIP”的扩展名将会被对待为 Open XML 文件。

LoadGrid 和 SaveGrid 方法同样拥有一些文件类型参数的重载方法。这将会允许你的控件根据文件格式来判断而不是依靠扩展名。举个例子，你的应用程序也许会使用 Open XML 格式文件来作为数据文件，但使用的是“XLSX”以外的其他扩展名。你可以指定文件的类型为“Excel”并使用 FileFlags 枚举选项中的 OpenXml 选项。

想要将表格保存和读取一个 OpenXml 文件，完成下面的步骤：

1. 在包含 C1FlexGrid 的窗体上添加三个按钮，并在他们的 Text 属性中输入 Save、Clear 和 Load。



2. 在设计器中，双击 Save（Button_1）按钮，并将下面的代码添加到 Button1_Click 事件中去：

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Me.C1FlexGrid1.SaveGrid("C:\test\myfile.xlsx",
FileFormatEnum.Excel, FileFlags.OpenXml)
End Sub
```

- C#

```
private void button1_Click(object sender, EventArgs e)
{
this.c1FlexGrid1.SaveGrid(@"C:\test\myfile.xlsx",
FileFormatEnum.Excel, FileFlags.OpenXml);
}
```

注意：你必须添加 **Imports** 或者 **using** 代码块到你的窗体代码的最上方，使得这段代码能够工作正常。如果编写 Visual Basic 请使用 **Imports C1.Win.C1FlexGrid**。如果编写 C#请使用 **using C1.Win.C1FlexGrid**;。

3. 请将下面的代码添加到 Button2_Click 事件中去，作为 Clear 按钮的内容：

- Visual Basic

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
Me.C1FlexGrid1.Clear(ClearFlags.All)
End Sub
```

- C#

```
private void button2_Click(object sender, EventArgs e)
{
this.c1FlexGrid1.Clear(ClearFlags.All);
}
```

4. 将下面的代码输入到 Button3_Click 事件中去，作为 Load 按钮的行为：

- Visual Basic

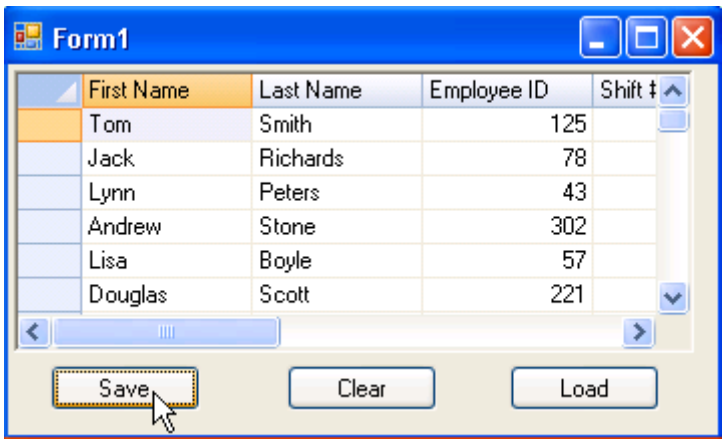
```
Private Sub button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
Me.C1FlexGrid1.LoadGrid("C:\test\myfile.xlsx",
FileFormatEnum.Excel, FileFlags.OpenXml)
End Sub
```

- C#

```
private void button3_Click(object sender, EventArgs e)
{
this.c1FlexGrid1.LoadGrid(@"C:\test\myfile.xlsx",
FileFormatEnum.Excel, FileFlags.OpenXml);
}
```

运行这个程序：

1. 在表格中输入你需要的内容。
2. 点击 Save 按钮。表格将会保存为一个 Open XML 文件，后缀名为.xlsx，文件将会保存到"C:\test"
3. 目录下。如果文件夹不存在的话，你必须创建这个文件夹。



4. 点击 **Clear** 按钮来清空表格。
5. 点击 **Load** 按钮，之前你保存的表格将会被重新读取出来。

7.20 用数据填充一个非绑定的表格

想要用数据填充非绑定表格，请使用表格的索引器来填充列、行或者一个单元格。要填充一个单元格区域，请使用 Data 属性。

7.20.1 用数据填充一列

想要用数据填充一列，在窗体加载的时候，用表格的索引设置一个循环来填充列。将下面的代码添加到 **Form_Load** 事件中，将 0 填充到第一列的所有行：

- Visual Basic

```
Dim r As Integer
For r = C1FlexGrid1.Rows.Fixed To C1FlexGrid1.Rows.Count - 1
Me.C1FlexGrid1(r, 1) = 0
Next
```

- C#

```
int r;
for (r = c1FlexGrid1.Rows.Fixed; r <= c1FlexGrid1.Rows.Count -1;r++)
{
this.c1FlexGrid1[r, 1] = 0;
}
```

本主题演示如下：

用 0 来填充第一列。

7.20.2 用数据填充一个单元格区域

想要用数据填充一个单元格区域，在窗体加载的时候，为 CellRange 设置 Data 属性。添加下面的代码到 **Form_Load** 事件中，设置 **Data** 属性为 0：

- Visual Basic

```
Dim rng As C1.Win.C1FlexGrid.CellRange = Me.C1FlexGrid1.GetCellRange(1,
1,
3, 3)
rng.Data = 0
```

- C#

```
C1.Win.C1FlexGrid.CellRange rng = this.c1FlexGrid1.GetCellRange(1,1,3,3);
rng.Data = 0;
```

本主题演示如下：

所有的单元格区域都被 0 填满了。

7.20.3 用数据填充一行

想要用数据填充一行，在窗体加载的时候，用表格的索引设置一个循环来填充行。添加下面的代码到 **Form_Load** 事件中，将第一行的所有列填充为 0：

- Visual Basic

```
Dim r As Integer
For r = C1FlexGrid1.Cols.Fixed To C1FlexGrid1.Cols.Count - 1
Me.C1FlexGrid1(1, r) = 0
Next
```

- C#

```
int r;
for (r = c1FlexGrid1.Cols.Fixed; r <= c1FlexGrid1.Cols.Count -1;r++)
{
this.c1FlexGrid1[1, r] = 0;
}
```

本主题演示如下：

第一行被 0 填满了。

7.20.4 用数据填充一个单元格

想要用数据填充一个单元格，在窗体加载的时候，用表格的索引来设置数据。添加下面的代码到 **Form_Load** 事件中， 设置单元格的文本：

- Visual Basic

```
Me.C1FlexGrid1(3, 2) = "Cell Text"
```

- C#

```
this.c1FlexGrid1[3, 2] = "Cell Text";
```

本主题演示如下：

文本 Cell Text 在第四行第三列。

7.21 限制表格的编辑

想要禁止编辑整个表格，特定行或者特定列，设置 **AllowEditing** 属性为 **False**。

相反地，要允许编辑的话，将 **AllowEditing** 属性设置为 **True**，这也是默认值。

7.21.1 在整个表格内禁止编辑

想要禁止整个表格内的编辑操作，可以把 AllowEditing 属性设置为 **False**，这个操作在设计器中或者在代码中都可以完成：

在设计器中：

在 C1FlexGrid 任务菜单中，取消 Enable Editing 的勾选框。

另外，在属性窗口中找到 **AllowEditing** 属性并将它设置为 **False**。

在代码中：

将下面的代码添加到 **Form_Load** 事件中去：

- Visual Basic

```
Me.C1FlexGrid1.AllowEditing = False
```

- C#

```
this.c1FlexGrid1.AllowEditing = false;
```

7.21.2 禁止某一列的编辑

想要禁止某一列的编辑操作，可以设置 AllowEditing 属性为 **False**，这个操作可以在设计器中或者在代码中完成：

在设计器中：

1. 在表格中选中你想编辑的一列。这将会打开这一列的**列任务菜单**。
2. 取消 Allow Editing 前的勾选框。

另外，AllowEditing 也可以通过 C1FlexGrid 列编辑器来设置：

1. 打开 **C1FlexGrid 列编辑器**。想要了解更多关于如何访问 **C1FlexGrid 列编辑器**的信息，请查看访问 C1FlexGrid 列编辑器 章节。
3. 在右侧的窗格中选中你希望编辑的列，并在左侧窗格中将这一列的 **AllowEditing** 属性设置为 **False** 。
4. 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 **Form_Load** 事件中来限制 AtomicMass 列的编辑操作：

- Visual Basic

```
Me.C1FlexGrid1.Cols("AtomicMass").AllowEditing = False
' 与这段代码相同: Me.C1FlexGrid1.Cols(4).AllowEditing = False
```

- C#

```
this.c1FlexGrid1.Cols["AtomicMass"].AllowEditing = false;
//与这段代码相同: this.c1FlexGrid1.Cols[4].AllowEditing = false;
```

7.21.3 禁止某一行的编辑

想要禁止某一列的编辑操作，可以设置 AllowEditing 属性为 **False**。在本例中，代码将会限制第六行的编辑：

- Visual Basic

```
Me.C1FlexGrid1.Rows(5).AllowEditing = False
```

- C#

```
this.c1FlexGrid1.Rows[5].AllowEditing = false;
```

7.22 限制一列的排序

想要限制一列的排序，可以在设计器中或者在代码中将 AllowSorting 属性设置为 **False**：

在设计器中：

1. 在表格中选择你想要编辑的一列。这将会打开这一列的**列任务菜单**。
2. 取消 Allow Sorting 前的勾选框。

另外， AllowSorting 属性也可以通过 **C1FlexGrid 列编辑器**来设置：

1. 打开 **C1FlexGrid 列编辑器**. 打开 **C1FlexGrid 列编辑器**。想要了解更多关于如何访问 **C1FlexGrid 列编辑器**的信息，请查看访问 C1FlexGrid 列编辑器章节（第 143 页）。
2. 在右侧的窗格中选中你希望编辑的列，并在左侧窗格中将这一列的 **AllowSorting** 属性设置为 **False**。
3. 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 **Form_Load** 事件中来限制 AtomicNumber 列的排序操作：

- Visual Basic

```
Me.C1FlexGrid1.Cols("AtomicNumber").AllowSorting = False
'与这段代码相同: Me.C1FlexGrid1.Cols(1).AllowEditing = False
```

- C#

```
this.c1FlexGrid1.Cols["AtomicNumber"].AllowSorting = false;
//与这段代码相同: this.c1FlexGrid1.Cols[1].AllowEditing = false;
```

7.23 缩放图像

当表格中的某一行的大小发生变换，想要在列中缩放图像，将 ImageAlign 属性设置为 **Scale**。

1. 将这一行的高度设置为第一行高度的两倍。将下面的代码添加到 Form_Load 事件中：

- Visual Basic

```
Me.C1FlexGrid1.Rows(1).Height = 2 * Me.C1FlexGrid1.Rows.DefaultSize
Me.C1FlexGrid1.Rows(2).Height = 2 * Me.C1FlexGrid1.Rows.DefaultSize
```

- C#

```
this.c1FlexGrid1.Rows[1].Height = 2 * this.c1FlexGrid1.Rows.DefaultSize;
this.c1FlexGrid1.Rows[2].Height = 2 * this.c1FlexGrid1.Rows.DefaultSize;
```

2. 在这一列中缩放图像。

在设计器中：

打开 C1FlexGrid 列编辑器. 打开 C1FlexGrid 列编辑器。想要了解更多关于如何访问 C1FlexGrid 列编辑器的信息，请查看访问 C1FlexGrid 列编辑器章节（第 143 页）。

在右侧窗格中选择一个包含有图像的列，然后在左侧窗格中将 ImageAlign 属性设置为 Scale。

点击 OK 来关闭编辑器。

代码中在：

将下面的代码添加到步骤 1 之后：

- Visual Basic

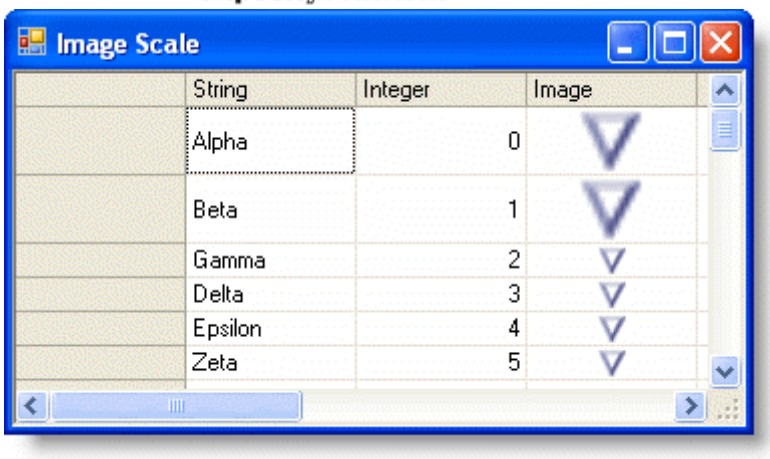
```
Me.C1FlexGrid1.Cols(3).ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.Scale
```

- C#

```
this.c1FlexGrid1.Cols[3].ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.Scale;
```

本主题演示如下：

通过使用 ImageAlign 属性，图像将被缩放，以适应单元格内的最大区域，同时保留图片的原始宽高比。



7.23.1 在整个表格中缩放图像

想要在整个表格中缩放图像，不只是一列中，请将 Normal 样式中的 ImageAlign 属性设置为 **Scale**。

在设计器中：

1. 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 **C1FlexGrid 样式编辑器**的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
2. 在**内**置样式列表中选择 Normal。
3. 在右侧窗格中，找到 ImageAlign 属性并将它设置为 Scale。
4. 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
Me.C1FlexGrid1.Styles("Normal").ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.Scale
```

- C#

```
this.c1FlexGrid1.Styles["Normal"].ImageAlign =
C1.Win.C1FlexGrid.ImageAlignEnum.Scale;
```

7.24 搜寻列中的条目

在用户输入中搜索需要的条目，请将 AutoSearch 属性设置为 **FromCursor** 来开始从当前行搜索，或者设置为 **FromTop** 来开始从第一个滚动的行搜索。相反，要想禁用搜索，将 AutoSearch 属性设置为 **None**，这也是默认值。这个属性可以在设计器或者在代码中设置。

在设计器中：

在属性窗口中找到 **AutoSearch** 属性并将它设置为 **FromTop**。

在代码中：

将下面的代码添加到 Form_Load 事件中来设置 AutoSearch 属性为 FromTop：

- Visual Basic

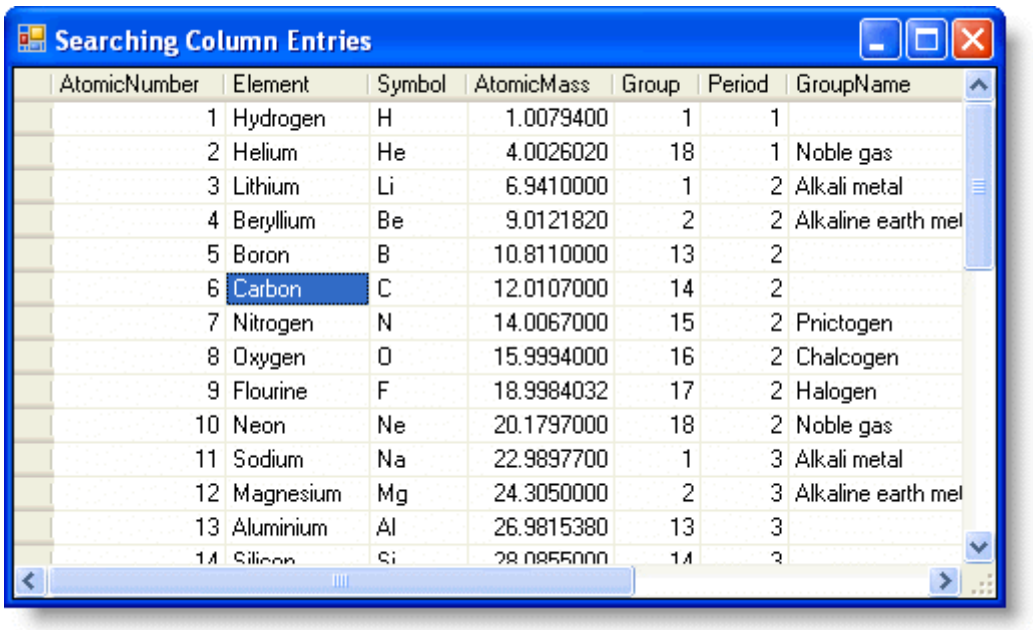
```
Me.C1FlexGrid1.AutoSearch = C1.Win.C1FlexGrid.AutoSearchEnum.FromTop
```

- C#

```
this.c1FlexGrid1.AutoSearch =
C1.Win.C1FlexGrid.AutoSearchEnum.FromTop;
```

本主题演示如下：

对于用户输入的搜索结果，将会高亮包含这个字母的单元格。在本例中，在 Element 列中输入 C 将会高亮 Carbon。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth mel
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth mel
13	Aluminium	Al	26.9815380	13	3	
14	Silicon	Si	28.0855000	14	3	

注意：如果多个条目以同一个字母开头，输入下个字母将会高亮包含这两个字母的条目。举个例子，在 Element 列中输入 He 将会高亮 Helium。

7.25 当用户按下 Delete 键时清空单元格

要想设置当用户按下 DELETE 键时清空单元格，请使用 **C1FlexGrid** 的 **KeyDown** 事件来获取 DELETE 键被按下的事件。

添加下面的 **KeyDown** 事件代码到你的窗体中

- Visual Basic

```
Private Sub C1FlexGrid1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles C1FlexGrid1.KeyDown
If (e.KeyCode = Keys.Delete) Then
C1FlexGrid1(C1FlexGrid1.Row, C1FlexGrid1.Col) = 0
End If
End Sub
```

- C#

```
private void c1FlexGrid1_KeyDown(object sender,
System.Windows.Forms.KeyDownEventArgs e)
{
if (e.KeyCode == Keys.Delete)
{
c1FlexGrid1(c1FlexGrid1.Row, c1FlexGrid1.Col) = 0;
}
}
```

7.26 将行设置为标题行

要将行设置为标题行，请为每一行设置 Caption 和 DataType 属性。

1. 将下面的代码添加到 Form_Load 事件中来设置将会出现在表格中的行数和列数。

- Visual Basic

```
Me.C1FlexGrid1.Cols.Count = 5
Me.C1FlexGrid1.Rows.Count = 7
```

- C#

```
this.c1FlexGrid1.Cols.Count = 5;
this.c1FlexGrid1.Rows.Count = 7;
```

2. 向 RowCollection 中添加行：

- Visual Basic

```
Dim row As C1.Win.C1FlexGrid.RowCollection = Me.C1FlexGrid1.Rows
```

- C#

```
C1.Win.C1FlexGrid.RowCollection row = this.c1FlexGrid1.Rows;
```

3. 为每一行设置 Caption 和 DataType 属性。

- Visual Basic

```
row(1).Caption = "Date"
row(1).DataType = GetType(DateTime)
row(2).Caption = "Contact"
row(2).DataType = GetType(String)
row(3).Caption = "Phone"
row(3).DataType = GetType(String)
row(3).EditMask = "(999)          999-9999;*"
row(4).Caption = "Platform"
row(4).DataType = GetType(String)
row(4).ComboList = "|Windows XP|Windows 2000|Windows ME|Windows
NT|Windows 98|Windows 95"
row(5).Caption = "Error Code"
row(5).DataType = GetType(Integer)
row(6).Caption = "Resolved"
row(6).DataType = GetType(Boolean)
```

- C#

```
row[1].Caption = "Date";
row[1].DataType = typeof(DateTime);
row[2].Caption = "Contact";
row[2].DataType = typeof(string);
row[3].Caption = "Phone";
row[3].DataType = typeof(string);
row[3].EditMask = "(999)          999-9999;*";
row[4].Caption = "Platform";
row[4].DataType = typeof(string);
row[4].ComboList = "|Windows XP|Windows 2000|Windows ME|Windows
NT|Windows 98|Windows 95";
row[5].Caption = "Error  Code";
row[5].DataType = typeof(int);
row[6].Caption = "Resolved";
row[6].DataType = typeof(bool);
```

4. 将标题行的字体格式化为 Tahoma, 9pt, Bold。

- Visual Basic

```
Me.C1FlexGrid1.Styles("Fixed").Font = New Font("Tahoma", 9,
FontStyle.Bold)
```

- C#

```
this.c1FlexGrid1.Styles["Fixed"].Font = new Font("Tahoma", 9,
FontStyle.Bold);
```

5. 合并固定行并添加标题行：

- Visual Basic

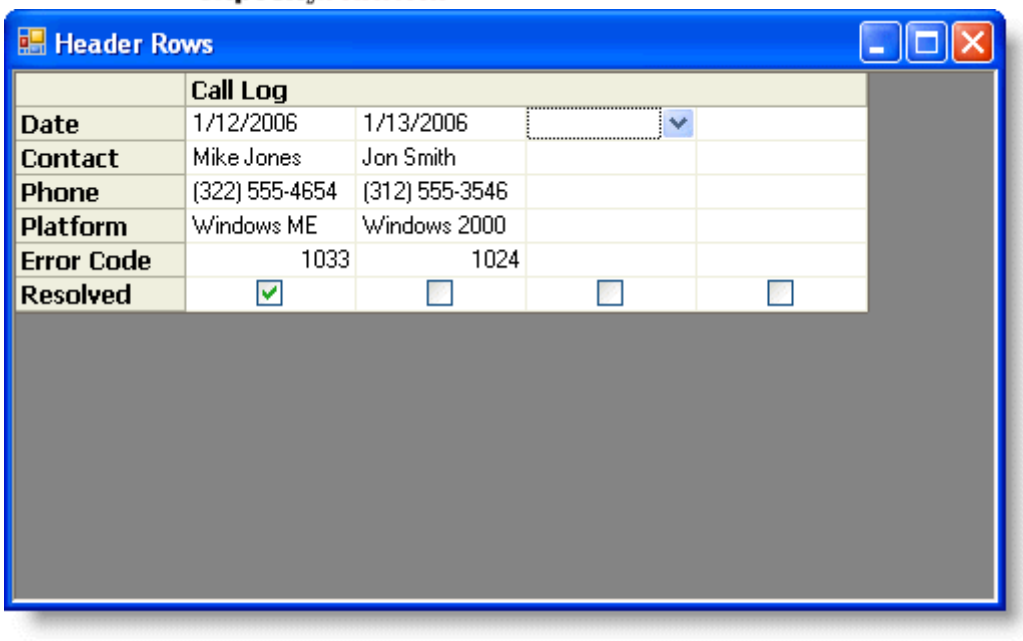
```
Me.C1FlexGrid1.AllowMerging =
C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
row(0).AllowMerging = True
Dim rng As C1.Win.C1FlexGrid.CellRange = C1FlexGrid1.GetCellRange(0, 1,
0, 4)
rng.Data = "Call Log"
```

- C#

```
this.c1FlexGrid1.AllowMerging =
C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
row[0].AllowMerging = true;
C1.Win.C1FlexGrid.CellRange rng = c1FlexGrid1.GetCellRange(0,1,0,4);
rng.Data = "Call Log";
```

本主题演示如下：

标题行将会出现在第一列并且每一行将会被格式化为它自己拥有的 DataType 属性：


Header Rows
Call Log
Date
1/12/2006
1/13/2006
Contact
Mike Jones
Jon Smith
Phone
(322) 555-4654
(312) 555-3546
Platform
Windows ME
Windows 2000
Error Code
1033
1024
Resolved

7.27 为行或者列设置背景色

要为列和行设置背景色，创建一个新的样式并将它指定到一行或者一列。

为列设置背景色

1. 为这一列创建一个新的样式。

在设计器中：

- 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 **C1FlexGrid 样式编辑器**的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
- 点击 **Add** 来创建一个新的样式。
- 双击 CustomStyle1 将它重命名为 ColumnColor，然后按下 ENTER。
- 不要退出 C1FlexGrid 样式编辑器.

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
Dim cc As C1.Win.C1FlexGrid.CellStyle
cc = Me.C1FlexGrid1.Styles.Add("ColumnColor")
```

- C#

```
C1.Win.C1FlexGrid.CellStyle cc =
this.c1FlexGrid1.Styles.Add("ColumnColor");
```

2. 将 BackColor 颜色设置为 CornSilk。

在设计器中：

- 在 C1FlexGrid 样式编辑器中，在右侧窗格中找到 BackColor 属性并将它设置为 CornSilk。
- 点击 OK 来关闭 C1FlexGrid 样式编辑器.

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
cc.BackColor = Color.Cornsilk
```

- C#

```
cc.BackColor = Color.Cornsilk;
```

3. 将下面的代码添加到 Form_Load 事件中，将这个样式指定到一列上：

- Visual Basic

```
Me.C1FlexGrid1.Cols(2).Style = Me.C1FlexGrid1.Styles("ColumnColor")
```

- C#

```
this.c1FlexGrid1.Cols[2].Style = this.c1FlexGrid1.Styles["ColumnColor"];
```

本主题演示如下：

Element 列的背景色将会被设置为 CornSilk。

为行设置背景色

1. 为这一行创建一个新的样式。

在设计器中：

- 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访**问 C1FlexGrid 样式编辑器**的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
- 点击 **Add** 来创建一个新的样式。
- 双击 CustomStyle1 将它重命名为 RowColor, 然后按下 ENTER。
- 不要退出 C1FlexGrid 样式编辑器.

在代码中：

将下面的代码添加到 **Form_Load** 事件中去：

- Visual Basic

```
Dim rs As C1.Win.C1FlexGrid.CellStyle
rs = Me.C1FlexGrid1.Styles.Add("RowColor")
```

- C#

```
C1.Win.C1FlexGrid.CellStyle rs =
this.c1FlexGrid1.Styles.Add("RowColor");
```

2. 将背景色设置为 PowderBlue.

在设计器中：

- 在 C1FlexGrid 样式编辑器中，找到 BackColor 属性并将它设置为 PowderBlue。
- 点击 OK 来关闭 C1FlexGrid 样式编辑器.

在代码中：

- Visual Basic

```
rs.BackColor = Color.PowderBlue
```

- C#

```
rs.BackColor = Color.PowderBlue;
```

2. 将下面的代码添加到 Form_Load 事件中，将这个样式指定到一行上：

- Visual Basic

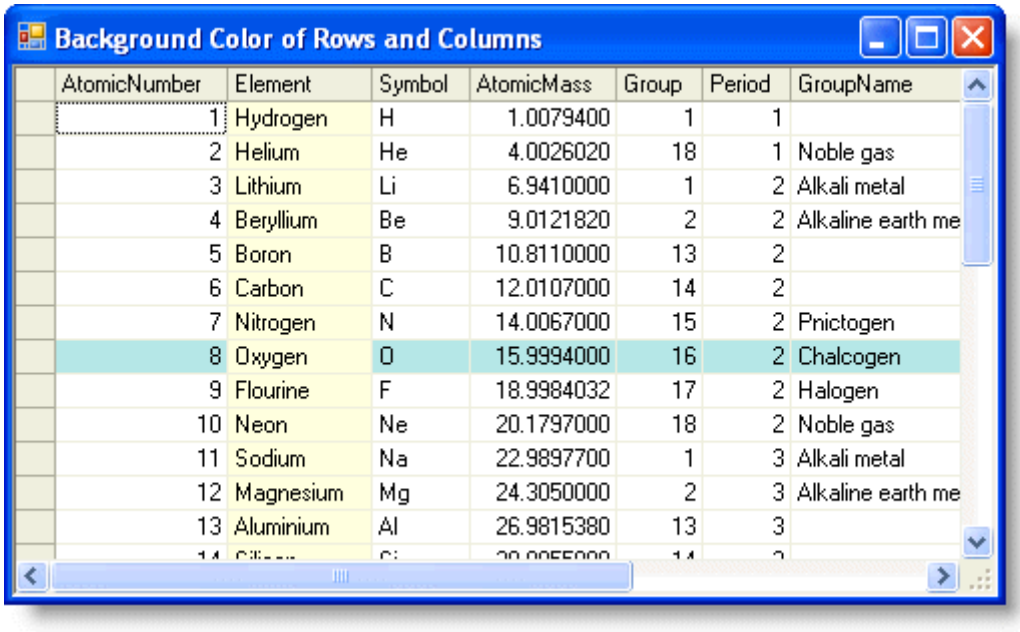
```
Me.C1FlexGrid1.Rows(8).Style = Me.C1FlexGrid1.Styles("RowColor")
```

- C#

```
this.c1FlexGrid1.Rows[8].Style = this.c1FlexGrid1.Styles["RowColor"];
```

本主题演示如下：

这一行的背景色将会被设置为 **PowderBlue**。注意，这个颜色将会替代该行本身的颜色。



AtomicNumber	Element	Symbol	AtomicMass	Group	Period	GroupName
1	Hydrogen	H	1.0079400	1	1	
2	Helium	He	4.0026020	18	1	Noble gas
3	Lithium	Li	6.9410000	1	2	Alkali metal
4	Beryllium	Be	9.0121820	2	2	Alkaline earth me
5	Boron	B	10.8110000	13	2	
6	Carbon	C	12.0107000	14	2	
7	Nitrogen	N	14.0067000	15	2	Pnictogen
8	Oxygen	O	15.9994000	16	2	Chalcogen
9	Flourine	F	18.9984032	17	2	Halogen
10	Neon	Ne	20.1797000	18	2	Noble gas
11	Sodium	Na	22.9897700	1	3	Alkali metal
12	Magnesium	Mg	24.3050000	2	3	Alkaline earth me
13	Aluminium	Al	26.9815380	13	3	

7.27.1 在一个语句中设置行和列的背景色

要使用一句代码来设置行和列的背景色，请将下面的代码添加到 **Form_Load** 事件中：

- Visual Basic

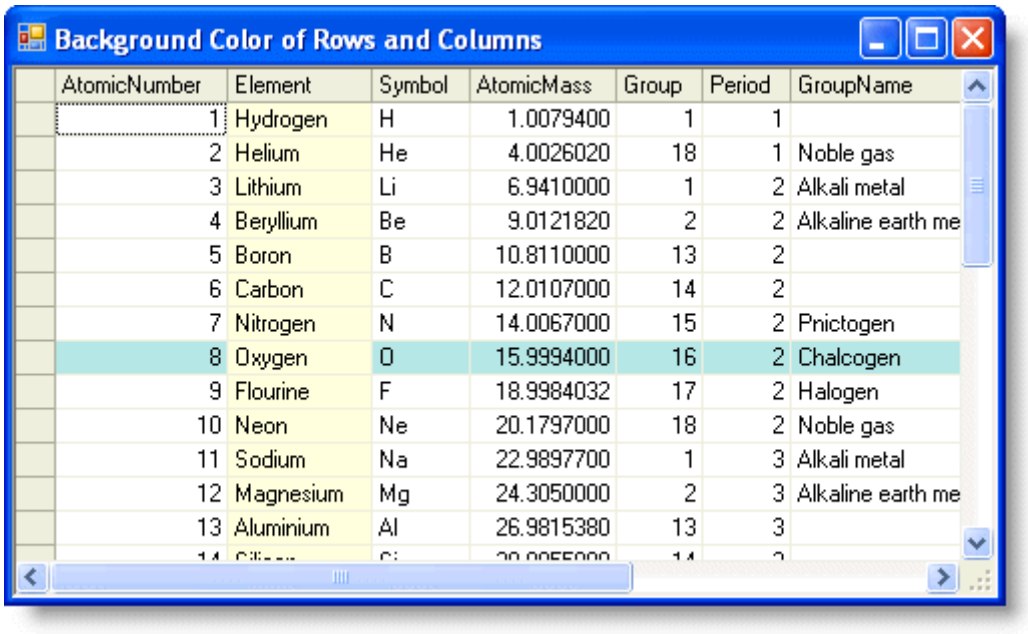
```
Me.C1FlexGrid1.Cols(2).StyleNew.BackColor = Color.Cornsilk
Me.C1FlexGrid1.Rows(8).StyleNew.BackColor = Color.PowderBlue
```

- C#

```
this.c1FlexGrid1.Cols[2].StyleNew.BackColor = Color.Cornsilk;
this.c1FlexGrid1.Rows[8].StyleNew.BackColor = Color.PowderBlue;
```

本主题演示如下：

这一行代码所带来的结果和创建一个新样式并指定到行上是一样的。然而，使用单独一句只改变它指定的那一行或者一列。要想将样式应用到多行或者多列，创建一个新样式并将它设置到列上或者行上，请看为行或者列设置背景色章节。



7.28 为单一单元格设置字体

要想为单一单元格设置字体，请创建一个新的样式并指定到一个单元格上。

1. 创建一个新样式。

在设计器中：

- 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 **C1FlexGrid 样式编辑器**的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
- 点击 **Add** 来创建一个新的样式。
- 双击 CustomStyle1 并将它重命名为 myStyle，完成后按下 ENTER 键。
- 不要退出 C1FlexGrid 样式编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
Dim cs As C1.Win.C1FlexGrid.CellStyle
cs = Me.C1FlexGrid1.Styles.Add("myStyle")
```

- C#

```
C1.Win.C1FlexGrid.CellStyle cs =
this.c1FlexGrid1.Styles.Add("myStyle");
```

2. 将字体设置为 Tahoma, 10 pt, Bold。

在设计器中：

- 在 **C1FlexGrid 样式编辑器**，在右侧窗格中找到 **Font** 属性并点击旁边的**省略号**按钮。Font 对话框将会弹出。
- 设置 Font 为 Tahoma。
- 设置 Font style 为 Bold。
- 设置 **Size** 为 **10**。
- 点击 **OK** 来关闭 **Font** 对话框。但是不要退出 **C1FlexGrid 样式编辑器**。

在代码中：

将下面的代码添加到 Form_Load 事件中去：

- Visual Basic

```
cs.Font = New Font("Tahoma", 10, FontStyle.Bold)
```

- C#

```
cs.Font = new Font("Tahoma", 10, FontStyle.Bold);
```

3. 将字体颜色设置为 Blue。

在设计器中：

- 在 C1FlexGrid 样式编辑器中，在右侧窗格中找到 ForeColor 属性并将它设置为 Blue。
- 点击 OK 来关闭 C1FlexGrid 样式编辑器.

在代码中：

将下面的代码添加到 **Form_Load** 事件中去：

- Visual Basic

```
cs.ForeColor = Color.Blue
```

- C#

```
cs.ForeColor = Color.Blue;
```

4. 将下面的代码添加到 Form_Load 事件中，来讲这个样式指定到一个单元格上：

- Visual Basic

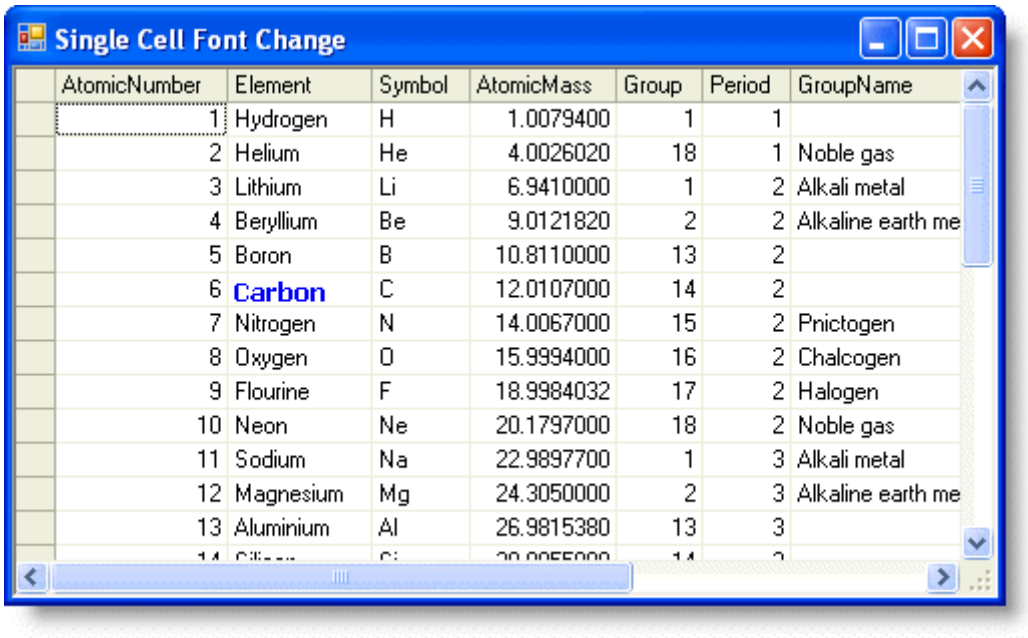
```
Me.C1FlexGrid1.SetCellStyle(6,  2, "myStyle")
```

- C#

```
this.c1FlexGrid1.SetCellStyle(6, 2, "myStyle");
```

本主题演示如下：

Carbon 将会显示为蓝色粗体 10 pt、Tahoma 字体。



7.29 在 C1FlexGrid 中设置文本分隔符

要在 C1FlexGrid 中设置文本分隔符，使用 **Split** 方法。

将下面的代码添加到 **Form_Load** 事件中来设置文本分隔符为分号：

- Visual Basic

```
Dim cols As String = "Product;Region;Salesperson;Sales;Bonus"
Dim colNames As String() = cols.Split(";")
Me.C1FlexGrid1.Cols.Count = 5
Me.C1FlexGrid1.Cols.Fixed = 0
Dim i%
For i = 0 To Me.C1FlexGrid1.Cols.Count - 1
Me.C1FlexGrid1(0, i) = colNames(i)
Me.C1FlexGrid1.Cols(i).Name = colNames(i)
Next
```

- C#

```
string cols = "Product;Region;Salesperson;Sales;Bonus";
string[] colNames = cols.Split(new char[] { ';' });
this.c1FlexGrid1.Cols.Count = 5;
this.c1FlexGrid1.Cols.Fixed = 0;
for (int i = 0; i <= this.c1FlexGrid1.Cols.Count - 1; i++)
{
this.c1FlexGrid1[0, i] = colNames[i];
this.c1FlexGrid1.Cols[i].Name = colNames[i];
}
```

本主题演示如下：

在 **Split** 中的字符串确定了分隔符。将字符串和 Split 方法中的分号换为逗号将会和使用分号有一样的效果。

7.30 多列排序

想要对多个列排序，将每一列的 Sort 属性都设置并使用 Sort 方法来对列进行排序。

1. 将下面的代码添加到 Form_Load 事件中来设置第二排序列为升序，第三排序列为降序。

- Visual Basic

```
Me.C1FlexGrid1.Cols(1).Sort = C1.Win.C1FlexGrid.SortFlags.Ascending
Me.C1FlexGrid1.Cols(2).Sort = C1.Win.C1FlexGrid.SortFlags.Descending
```

- C#

```
this.c1FlexGrid1.Cols[1].Sort = C1.Win.C1FlexGrid.SortFlags.Ascending;
this.c1FlexGrid1.Cols[2].Sort = C1.Win.C1FlexGrid.SortFlags.Descending;
```

2. 将下面的代码添加到 Sort 方法中来进行第二和第三列的升序排列。

- Visual Basic

```
Me.C1FlexGrid1.Sort(C1.Win.C1FlexGrid.SortFlags.UseColSort,          1,
  2)
```

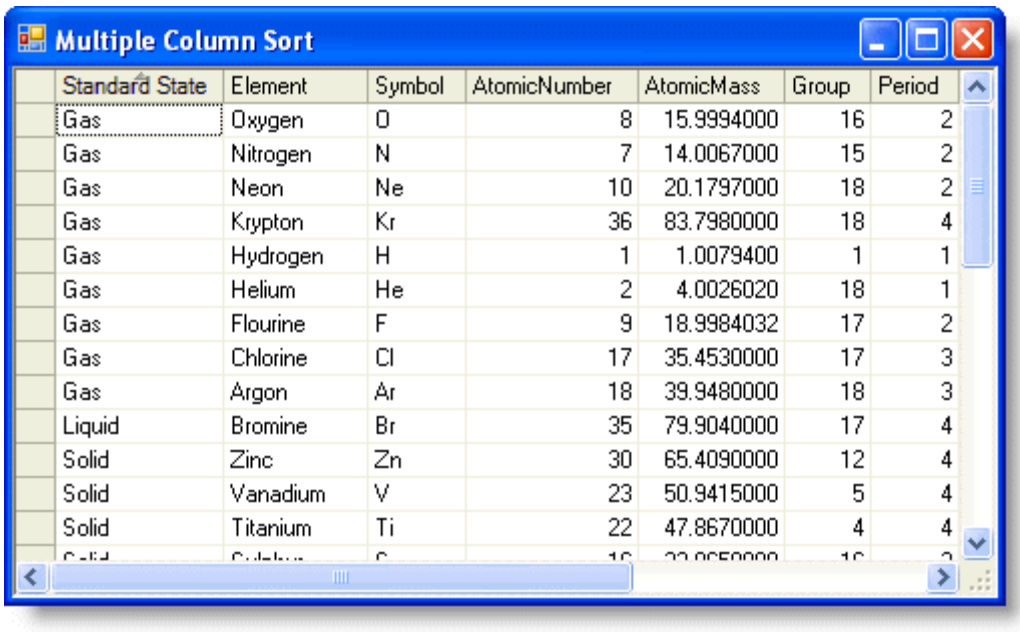
- C#

```
this.c1FlexGrid1.Sort(C1.Win.C1FlexGrid.SortFlags.UseColSort, 1, 2);
```

本主题演示如下：

你的表格将会看起来如下图所示，第二列为升序排列，第三列为降序排

列。在本例中，表格中的的 StandardState 列如果升序排列 Element 列如果降序排列的话，Element 列中的 Neon 将会出现在 Argon 前面。



Standard State	Element	Symbol	AtomicNumber	AtomicMass	Group	Period
Gas	Oxygen	O	8	15.9994000	16	2
Gas	Nitrogen	N	7	14.0067000	15	2
Gas	Neon	Ne	10	20.1797000	18	2
Gas	Krypton	Kr	36	83.7980000	18	4
Gas	Hydrogen	H	1	1.0079400	1	1
Gas	Helium	He	2	4.0026020	18	1
Gas	Flourine	F	9	18.9984032	17	2
Gas	Chlorine	Cl	17	35.4530000	17	3
Gas	Argon	Ar	18	39.9480000	18	3
Liquid	Bromine	Br	35	79.9040000	17	4
Solid	Zinc	Zn	30	65.4090000	12	4
Solid	Vanadium	V	23	50.9415000	5	4
Solid	Titanium	Ti	22	47.8670000	4	4

7.31 取消排序

要在 C1FlexGrid 中取消排序，当这个表格绑定到一个 DataTable 时，将 **DefaultView** 属性设置为 null。

将下面的代码添加到 **Button1_Click** 事件中：

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
CustTable.DefaultView.Sort = ""
End Sub
```

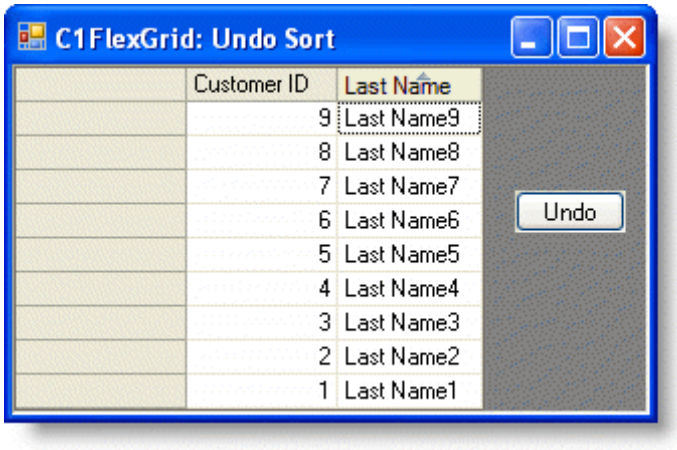
- C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
CustTable.DefaultView.Sort = "";
}
```

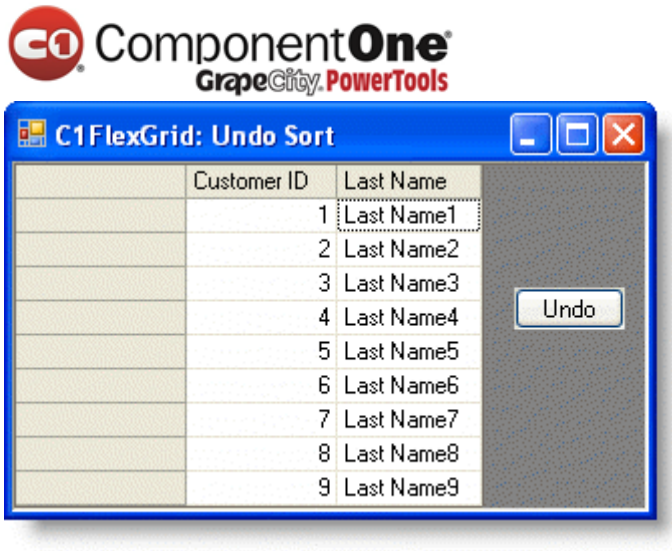
注意：DataTable.**DefaultView** 将会返回 DataTable 中的 DataView，并将排序字段设置为 null 来强制 DataView 取消之前的排序。

本主题演示如下：

点击 Last Name 来排序。



点击 **Undo** 按钮，排序将会被取消。


C1FlexGrid: Undo Sort
Customer ID
Last Name
1 Last Name1
2 Last Name2
3 Last Name3
4 Last Name4
5 Last Name5
6 Last Name6
7 Last Name7
8 Last Name8
9 Last Name9
Undo

7.32 在 C1FlexGrid 中使用密码项

要在单元格中输入密码时显示占位符（*），请使用 SetupEditor 事件。

1. 在表格中创建一列密码列并设置它的绘制模式：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Me.C1FlexGrid1.Cols(0).Width = Me.C1FlexGrid1.Rows(0).HeightDisplay
Me.C1FlexGrid1.ShowCursor = True
Me.C1FlexGrid1.Cols(1).Caption =((Me.C1FlexGrid1.Cols(1).Name) =
"Password")
Me.C1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
End Sub
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
this.c1FlexGrid1.Cols[0].Width =
this.c1FlexGrid1.Rows[0].HeightDisplay;
this.c1FlexGrid1.ShowCursor = true;
this.c1FlexGrid1.Cols[1].Caption = this.c1FlexGrid1.Cols[1].Name =
"Password";
this.c1FlexGrid1.DrawMode =
C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
}
```

2. 将下面的代码添加到 SetupEditor 事件中。这段代码将用户输入的字母隐藏起来。

- Visual Basic

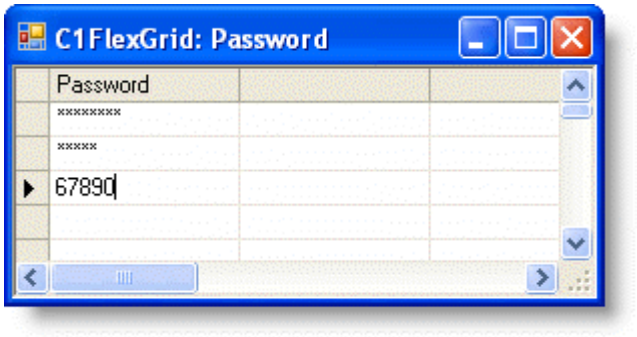
```
Private Sub C1FlexGrid1_SetupEditor(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles C1FlexGrid1.SetupEditor
Dim tb As TextBox = Me.C1FlexGrid1.Editor
If Not (tb Is Nothing) Then
If Me.C1FlexGrid1.Cols(e.Col).Name = "Password" Then
tb.PasswordChar = "*"c
Else
tb.PasswordChar = CChar(0)
End If
End If
End Sub
```

- C#

```
private void c1FlexGrid1_SetupEditor(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
TextBox tb = this.c1FlexGrid1.Editor as TextBox;
if (tb != null)
{
if (this.c1FlexGrid1.Cols[e.Col].Name == "Password")
tb.PasswordChar = '*';
else
tb.PasswordChar = (char)0;
}
}
```

本主题演示如下：

当用户在 Password 列中键入密码时之后并按下 ENTER，文本内容将会自动的被隐藏起来。



7.32.1 隐藏已经输入的字符

要想隐藏已经输入并且不需要编辑的文字，请使用 OwnerDrawCell 事件。

1. 将下面的代码添加到 **OwnerDrawCell** 事件中。这段代码将会隐藏已经输入并且不需要编辑的文字。

- Visual Basic

```
Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e
As C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles
C1FlexGrid1.OwnerDrawCell
If e.Row >= Me.C1FlexGrid1.Rows.Fixed And
Me.C1FlexGrid1.Cols(e.Col).Name = "Password" Then
e.Text = New String("*"c, e.Text.Length)
End If
End Sub
```

- C#

```
private void c1FlexGrid1_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
if (e.Row >= this.c1FlexGrid1.Rows.Fixed &&
this.c1FlexGrid1.Cols[e.Col].Name == "Password")
{
e.Text = new string('*', e.Text.Length);
}
}
```

2. 将下面的代码添加到 Form_Load 事件中

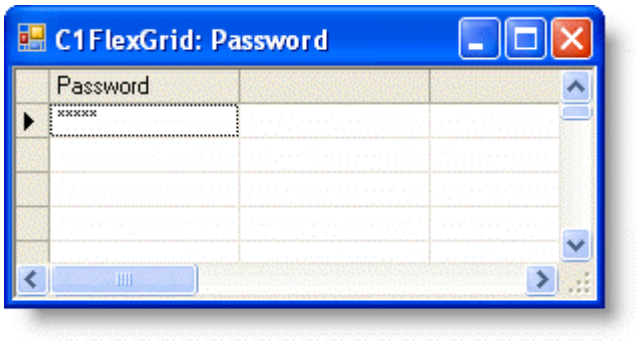
- Visual Basic

- C#

```
this.c1FlexGrid1[1,1] = "123456";
```

本主题演示如下：

再次运行程序，请注意 Password 列中自动填充到窗体上的的数字被变为星号。



7.33 在列头或者固定行换行

要想在列头或者固定行换行，请设置 Height 和 WordWrap 属性。

1. 为列头设置 Caption 属性

在设计器中：

- 在表格中选择一列，这将会打开这一列的**列任务菜单**。
- 在 Column Caption 框中，输入 Word Wrapping in Header。

另外， Caption 属性也可以在 **C1FlexGrid 列编辑器**中设置。

- 打开 **C1FlexGrid 列编辑器**. 打开 C1FlexGrid 列编辑器。想要了解更多关于如何访问 **C1FlexGrid 列编辑器**的信息，请查看访问 C1FlexGrid 列编辑器章节（第 143 页）。
- 在右侧窗格中选择一列并在左侧窗格中将它的 **Caption** 属性设置为 **Word Wrapping in Header**。
- 点击 OK 来关闭编辑器。

在代码中：

将下面的代码添加到 Form_Load 事件中。

- Visual Basic

```
Me.C1FlexGrid1.Cols(1).Caption = "Word Wrapping in Header"
```

- C#

```
this.c1FlexGrid1.Cols[1].Caption = "Word Wrapping in Header";
```

2. 设置列头的高度。

- Visual Basic

```
Me.C1FlexGrid1.Rows(0).Height = 3 * Me.C1FlexGrid1.Rows.DefaultSize
```

- C#

```
this.c1FlexGrid1.Rows[0].Height = 3 * this.c1FlexGrid1.Rows.DefaultSize;
```

3. 允许固定单元格的换行。

在设计器中：

- 打开 **C1FlexGrid 样式编辑器**。想要了解更多关于如何访问 **C1FlexGrid 样式编辑器**的信息，请查看访问 C1FlexGrid 样式编辑器 章节（第 143 页）。
- 在内置样式列表中选择 **Fixed**。
- 在右侧窗格中找到 **WordWrap** 属性并将它设置为 **True**。

- 点击 **OK** 来关闭设计器。

在代码中：

将下面的代码添加到 **Form_Load** 事件中去：

- Visual Basic

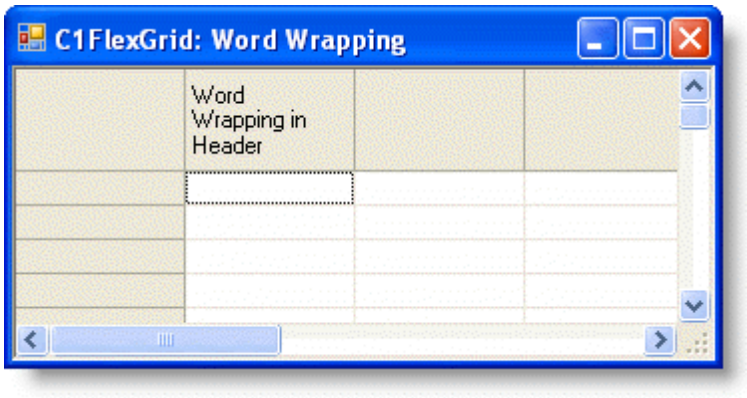
```
    Me.C1FlexGrid1.Styles("Fixed").WordWrap = True
```

- C#

```
    this.c1FlexGrid1.Styles["Fixed"].WordWrap = true;
```

本主题演示如下：

在本例中，一个三行的列头被创建了，文字是自动折行的。



8. WinForms 中的 FlexGrid 关键技巧

下面的一些技巧是由 C1FlexGrid 论坛中总结来的一些常见问题。

技巧 1：请使用 BeginUpdate/EndUpdate 方法来提升性能

每次当一个单元格的值发生更改时，或多行、多列被添加或者从表格中删除，一些计算被执行用来重新计算表格的布局和更新显示。你可以通过调用 BeginUpdate 和 EndUpdate 方法封闭这些变化来显著提高性能。例如：

- Visual Basic

```
' 在更新表格前调用 BeginUpdate。
flex.BeginUpdate()
' 做修改。
Try
Dim r As Integer = 0
Do While (r < _flex.Rows.Count)
Dim c As Integer = 0
Do While (c < _flex.Cols.Count)
_flex(r, c) = (r + c)
c += 1
Loop
r += 1
Loop
Finally
' 完成后调用 EndUpdate 。
_flex.EndUpdate()
End Try
```

- C#

```
//在更新表格前调用 BeginUpdate。
flex.BeginUpdate();
//做修改。
try
{
for (int r = 0; r < _flex.Rows.Count; r++)
{
For (int c = 0; c < _flex.Cols.Count; c++)
{
_flex[r, c] = r + c;
}
}
}
finally
{
//完成后调用 EndUpdate 。
_flex.EndUpdate();
}
```

注意，使用 **try/finally** 块来确保 C1FlexGridBase.EndUpdate 方法被调用，即使更新代码失败抛出异常。

注意：BeginUpdate 和 EndUpdate 方法在 C1FlexGrid 2010/v1 release 版本中添加。在之前的版本中，Redraw 属性被用来达到同样的目的。这种变化是用来增加和其他控件使用 BeginUpdate 和 EndUpdate 模式的兼容性。

技巧 2：使用 AutoResize 属性来提升性能

当一个绑定的表格从一个数据源检索数据，它可以测量每个单元格，并设置列宽使它们适合所有的数据。这将确保表格的布局是显示数据源的最佳布局，但如果数据源很大时，它很耗费时间（如果超过上千行的话）。

基于这种情况，你应该将 AutoResize 属性设置为 False 并且使用代码来控制列宽。

注意：从 C1FlexGrid 2010/v1 release 版本开始，AutoResize 属性默认为 False。如果你的数据源有比较少的数据行数，并且你希望表格自动控制列宽的话，你可以将 AutoResize 属性设置为 True 或者在绑定、手动填充数据之后调用 AutoSizeCols 方法。这个改变适用于提高绑定大型数据源的性能。

技巧 3：基于单元格的值使用 DrawMode 属性动态指定单元格样式

表格允许你创建单元格样式并将它指定到行、列和任意单元格范围。你可以使用这个功能来根据单元格内容格式化单元格。例如，包含负值的单元格可以显示为红色。

你可以通过使用 SetCellStyle 方法来指定样式到单元格上做到这一点，但是在这个例子中，当单元格值发生变化时随时更新这个样式。同样的，如果表格绑定到数据源的话，在数据源发生重置时样式会丢失（如排序和过滤操作执行之后）。

在这种情况下，一个好的替代方案是使用表格的 OwnerDraw 功能来动态选择样式，当然是基于单元格的值。

例如，下面的代码展示了负值为红色，超过 1000 的为绿色：

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
' 使用随机值填充一列。
_flex.Cols(1).DataType = GetType(Integer)
Dim rnd As New Random
Dim r As Integer = 1
Do While (r < _flex.Rows.Count)
_flex(r, 1) = rnd.Next(-10000, 10000)
r += 1
Loop
'  创建一个样式来显示负值。
_flex.Styles.Add("Red").ForeColor = Color.Red
'  创建一个样式来显示值>= 1000。
_flex.Styles.Add("Green").ForeColor = Color.Green
' 通过设置 DrawMode 属性来允许 OwnerDraw。
_flex.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
End Sub
Private Sub _flex_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles _flex.OwnerDrawCell
' 检查行或者列包含整数数据。
If ((e.Row > 0) AndAlso (_flex.Cols(e.Col).DataType Is
GetType(Integer))) Then
' 拿到将要绘制的单元格中的值。
Dim value As Integer = CInt(_flex(e.Row, e.Col))
If (value < 0) Then
' 如果单元格值< 0，使用 Red 样式。
e.Style = _flex.Styles("Red")
ElseIf (value >= 1000) Then
' 如果单元格值>= 1000，使用 Green 样式。
e.Style = _flex.Styles("Green")
End If
End If
End Sub
```

- C#

```
private void Form1_Load(object sender, EventArgs e)
{
//使用随机值填充一列。
_flex.Cols[1].DataType = typeof(int);
Random rnd = new Random();
for (int r = 1; r < _flex.Rows.Count; r++)
{
_flex[r, 1] = rnd.Next(-10000, 10000);
}
//创建一个样式来显示负值。
_flex.Styles.Add("Red").ForeColor = Color.Red;
//创建一个样式来显示值>= 1000。
_flex.Styles.Add("Green").ForeColor = Color.Green;
//通过设置 DrawMode 属性来允许 OwnerDraw。
_flex.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
_flex.OwnerDrawCell += new
C1.Win.C1FlexGrid.OwnerDrawCellEventHandler(_flex_OwnerDrawCell);
}
private void _flex_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
//检查行或者列包含整数数据。
if (e.Row > 0 && _flex.Cols[e.Col].DataType == typeof(int))
{
//拿到将要绘制的单元格中的值。
int value = (int)_flex[e.Row, e.Col];
if (value < 0)
{
//如果单元格值< 0，使用 Red 样式。
e.Style = _flex.Styles["Red"];
}
else if (value >= 1000)
{
//如果单元格值>= 1000，使用 Green 样式。
e.Style = _flex.Styles["Green"];
}
}
}
```

技巧 4：不要在 OwnerDrawCell 事件中更改样式

注意上面的技巧 3 中的代码并没有在 OwnerDrawCell 事件中修改作为参数传递进来的 CellStyle 对象。反而，它为 e.Style 参数指定了一个新的值。

这是非常重要的，因为将 CellStyle 作为参数在事件处理中传递被其他单元格经常使用。例如，你可能在无意间改变了表格中大多数单元格使用的正常样式。

下面的例子说明了他们之间的区别：

- Visual Basic

```
' ** 正确的做法：
Private Sub _flex_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles _flex.OwnerDrawCell
' 当绘制这个单元格式选择使用的样式：
e.Style = MyStyleSelector(e.Row, e.Col)
End Sub
```

- C#

```
// ** 正确的做法：
private void _flex_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
//当绘制这个单元格式选择使用的样式：
e.Style = MyStyleSelector(e.Row, e.Col);
}
```

对比下面这个：

- Visual Basic

```
' ** 错误的做法：
Private Sub _flex_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs) Handles _flex.OwnerDrawCell
' 当绘制这个单元格式选择使用的样式：
' 这是不好的，因为改变任何 CellStyle 对象会导致表格失效，
' 而且将会引起这个事件处理方法被一次又一次的调用。
e.Style.Color = MyColorSelector(e.Row, e.Col)
End Sub
```

- C#

技巧 5：使用 Trimming 属性来在表格的列上显示省略号

```
// ** 错误的做法：
private void _flex_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
// 当绘制这个单元格式选择使用的样式：
// 这是不好的，因为改变任何 CellStyle 对象会导致表格失效，
// 而且将会引起这个事件处理方法被一次又一次的调用。
e.Style.Color = MyColorSelector(e.Row, e.Col);
}
```

Trimming 属性应该被使用来在表格的一列上显示省略号。要决定如何裁剪长字符串以适应单元格，Trimming 属性可以被设置为 **None**、**Character**、**Word**、**EllipsisCharacter**、**EllipsisWord**、或 **EllipsisPath**。

下面的表格描述了每个选项：

选项名称	描述
Character	以最接近的字符对指定的文本进行裁剪。
EllipsisCharacte	以最接近的字符对指定的文本进行裁剪，并在最后加上省略号。
EllipsisPath	文字的中心的一部分将会被移除并被省略号代替。这个算法将会尽可能多的保留斜杠分割中的部分。
EllipsisWord	指定的文本被裁剪最近的单词，并在结尾处插入省略号。
None	不进行任何裁剪。
Word	按最近的单次进行裁剪

下面的代码设置 Trimming 属性，用来在第二列上的最后展示省略号，并且将文本内容裁剪为最近的字符：

- Visual Basic

```
_flex.Cols(1).StyleNew.Trimming = StringTrimming.EllipsisCharacter
```

- C#

```
_flex.Cols[1].StyleNew.Trimming =StringTrimming.EllipsisCharacter;
```

技巧 6：使用 WordWrap 属性来在单元格中显示多行文本

当你想在一个单元格中显示多行文本时，请使用 WordWrap 和 Height 属性。WordWrap 属性决定表格是否应该自动断开包含空格的长字符串，并将它显示为多行文本。如果字符串中包含硬换行符（vbCrLf 或者"\n\r"），将会一直显示为多行文本。

多行文本可以显示在固定、可滚动的单元格中。例如，在固定单元格中设置多行文本，请查看在列头或者固定行换行章节（第 202 页）。

下面的代码展示了在可滚动的单元格中设置一个多行显示文本：

- Visual Basic

```
'设置 WordWrap 属性。
_flex.Styles("Normal").WordWrap = True
'设置行高。
_flex.Rows(1).Height = 2 *        fg.Rows.DefaultSize
'向单元格中添加文本。
_flex(1, 2) = "This is the first line." & ControlChars.CrLf & " This is
the second line."
```

- C#

```
// 设置 WordWrap 属性。
_flex.Styles["Normal"].WordWrap = true;
// 设置行高。
_flex.Rows[1].Height = 2 *        fg.Rows.DefaultSize;
// 向单元格中添加文本。
_flex[1, 2] = "This is the first line. \r\n This is the second line.";
```

技巧 7：当绑定到一个 DataTable 时使用 Sort 属性来检索数据进行排序

如果表格绑定到一个 DataTable，当数据刷新时，用户可以保持该排序方式。这可以通过使用默认视图的 Sort 属性和排序表达式。Sort 属性使用一个字符串包含列名的 ASC（对列进行升序排列）或者 DESC（对列进行降序排列）。默认情况下，使用升序排列。

多个列的排序可以通过输入每列的列名，中间使用逗号分隔。

一个排序表达式可以包含表格中的列名或者一个计算式。在运行时设置排序表达式，可以及时的反应数据视图中的变化。

-

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Me.ProductsTableAdapter.Fill(Me.NwindDataSet.Products)
End Sub
Private Sub btn_Sort_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Sort.Click
' 先根据 UnitsInStock 列进行排序，之后根据 ProductID 列进行排序。
Me.ProductsBindingSource.Sort = "UnitsInStock ASC, ProductID ASC"
End Sub
Private Sub btn_ClearSort_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_ClearSort.Click
' 清除排序
Me.ProductsBindingSource.Sort = ""
End Sub
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
this.productsTableAdapter.Fill(this.nwindDataSet.Products);
}
private void btn_Sort_Click(object sender, EventArgs e)
{
//先根据 UnitsInStock 列进行排序，之后根据 ProductID 列进行排序。
this.productsBindingSource.Sort = "UnitsInStock ASC, ProductID ASC";
}
private void btn_ClearSort_Click(object sender, EventArgs e)
{
//清除排序
this.productsBindingSource.Sort = "";
}
```

技巧 8：使用 SetupEditor 事件来控制在列中输入的字符个数

要在给定的列中，设置用户可以输入的最大字符个数，请使用 SetupEditor 事件。例如，一个 C1TextBox 被设置为 C1FlexGrid 中的编辑器，你必须在 C1FlexGrid 的 StartEdit 中声明使用一个外部编辑器：

-

- Visual Basic

```
Private Sub _flex_StartEdit(ByVal sender As System.Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles _flex.StartEdit
_flex.Editor = C1TextBox
End Sub
```

- C#

```
private void _flex_StartEdit(object sender,
C1.Win.C1FlexGrid.RowColEventArgs e)
{
_flex.Editor = c1TextBox;
}
```

现在，你设置了表格中的编辑器，你可以使用下面的代码来允许第三列最多输入 20 个字符，其他列最多输入 10 个字符（请记住下面的代码必须放到 SetupEditor 事件中）：

- Visual Basic

```
Private Sub _flex_SetupEditor(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs) Handles _flex.SetupEditor
' 设置第三列最多允许输入 20 个字符其他列最多 10 个。
If e.Col = 2 Then
CType(fg.Editor,  C1TextBox).MaxLength = 20
Else
CType(fg.Editor,  C1TextBox).MaxLength = 10
End If
End Sub
```

- C#

```
private void _flex_SetupEditor(object sender, RowColEventArgs e)
{
// 设置第三列最多允许输入 20 个字符其他列最多 10 个。
if (e.Col == 2)
c1TextBox.MaxLength = 20;
else
c1TextBox.MaxLength = 10;
}
```