使用FlexGrid

Silverlight及WPF版C1FlexGrid控件在相关的ComponentOne StudioV2 2010发布版中引入，自那时以来，该控件已经得到大幅改善。

选择

选择和选择模式

除了以表格形式显示数据外，大多数grid控件允许用户使用鼠标和键盘选择数据的一部分。
C1FlexGrid具有丰富的选择模型，由SelectionMode属性进行控制。以下选择模式可用： Cell：选择对应单个单元格。
CellRange：选择对应于一个单元格区域（相邻单元格的一块区域）。
Row：选择对应于一个整行。
RowRange：选择对应于一段连续的整行。
ListBox：
选择对应于一组行（不一定是连续的）。默认的SelectionMode是CellRange，提供了一个用户熟悉的，类似于Excel操作风格的选择行为。基于行进行选择的选项在需要选择整个数据行而不是单个Grid单元格的场景下同样有用。
无论当前的选择模式是什么，Grid都通过Selection属性暴露了当前的选择。此属性获取或设置当前的选择作为一个
CellRange的对象。

监控选择

一旦选择发生变化，可以是由于用户操作或者代码修改所引起，Grid将触发SelectionChanged事件，允许您处理新的选择结果。
例如，下面的代码可以监视选择并在选择更改时发送信息到控制台： C#

```
void _flex_SelectionChanged(object sender, CellRangeEventArgs e)
{
CellRange sel = _flex.Selection;
Console.WriteLine("selection: {0},{1} - {2},{3}", sel.Row, sel.Column, sel.Row2, sel.Column2);
Console.WriteLine("selection content: {0}",
GetClipString(_flex, sel));
}
static string GetClipString(C1FlexGrid fg, CellRange sel)
{
var sb = new System.Text.StringBuilder(); for (int r = sel.TopRow; r <= sel.BottomRow; r++)
{
for (int c = sel.LeftColumn; c <= sel.RightColumn; c++)
{
sb.AppendFormat("{0}\t"

, fg[r, c].ToString());


}
sb.AppendLine();
}
return sb.ToString(); }
```

一旦选择发生变化，代码将列举表示当前选择的CellRange的坐标。
同时它还使用GetClipString方法输出选择范围的内容，该方法遍历选择的项目并使用Grid的索引获取选择范围中每一个单元格的内容。关于索引，在本文档之前有所介绍。
请注意，GetClipString方法中的for循环使用单元格的TopRow，ButtomRow，LeftColumn以及RightColumn苏醒，而不是Row，Row2，Column以及Column2属性。这是必须的，因为Row可能大于或者小于Row2，取决于用户如何执行该选择（在选择时向上或者向下拖拽鼠标）。您可以很容易地通过Rows.GetDataItems方法从Selection中抽取大量有用的信息。该方法返回一个关联到一个
CellRange的数据项的集合。一旦你有了这个集合，你可以使用LINQ进行提炼和总结有关选定项目的信息。例如，可以考虑为一个绑定到Customer对象集合的Grid使用以下SelectionChanged事件处理的替代方案： C#

```
void _flex_SelectionChanged(object sender, CellRangeEventArgs e)
{
// 获取选定范围内的客户对象
var customers =
_flex.Rows.GetDataItems(_flex.Selection).OfType<Customer>();
// 使用LINQ从选中的客户对象集合中抽取信息
_lblSelState.Text = string.Format(
"{0} items selected, {1} active, total weight: {2:n2}", customers.Count(),
(from c in customers where c.Active select c).Count(),
(from c in customers select c.Weight).Sum()); }
```

注意，该代码使用OfType运算符将选中的数据项转换为Customer类型。一旦该步骤完成，代码将使用LINQ获取一个总数，一个"active"状态的客户数，以及选中客户的总权重。LINQ是完成这类工作的完美工具。它是灵活，富有表现力的，紧凑而且高效。

通过代码选择单元格和对象

Selection属性是可读写的，所以你可以使用代码选择单元格范围。您还可以使用Select方法进行选择，就像在WinForms版本的C1FlexGrid控件那样。Select方法允许您选择单元格或范围，并可选地滚动新的选择范围到可视视图，以便用户可以看到它。例如，为了选中Grid的第一个单元格并确保它位于用户可见范围，应使用以下代码：

C#

```
// 选择第零行，第零列，并确保该单元格可见 fg.Select(0, 0, true);
```

选择相关的方法均基于行和列的索引进行工作。但是您也可以用来基于单元格内容进行选择。例如，以下代码选择在Grid的"Name"列中，第一个包含指定字符串的行。 C#

```
bool SelectName(string name)
{
// 在"Name"列查找包含指定字符串的行
int

col = _flexGroup.Columns[

"Name"].Index;
int row = FindRow(_flex, name, _flex.Selection.Row, col, true); if (row > -1)
{
_flex.Select(row, col); return true;
}
// 未查询到……
return false; }
```

该代码使用的FindRow辅助方法定义如下：

C#

```
// 在一个指定的列中查找包含某个文本的行
int FindRow(C1FlexGrid flex, string text, int startRow, int col, bool wrap)
{
int count = flex.Rows.Count;
for (int off = 0; off <= count; off++)
{
// 搜索到底部且不允许循环查找？立即退出
if (!wrap && startRow + off >= count)
{ break; }
// 从行中获取文本
int row = (startRow + off) % count; var

content = flex[row, col];


// 如果查找到匹配项则返回匹配行的索引
if (content != null &&
content.ToString().IndexOf(text,
StringComparison.OrdinalIgnoreCase) > -1)
{
return row;
}
}
// 未查询到…… return -1; }
```

该FindRow方法实现了在C1FlexGrid的WinForms版本中提供的FindRow方法的功能。此方法没有内置在Silverlight或WPF版本的Grid产品中，以使得控件所占用的尺寸尽可能的小。该方法在给定的列中搜索指定的字符串，从指定的行开始搜索，并可选地在搜索到末尾仍未查找到时返回到行首开始搜索。这在许多场景下已经具有足够的使用灵活性。

另一个常见的选择场景是在数据源中选择一个特定对象的情况。你的第一反应可能会查找并使用PagedCollectionView.IndexOf 方法从数据源集合中查找对象的索引，然后使用索引选择行。这种方法的问题是，它只有在数据没有进行分组时能够正常工作。如果已经将数据进行了分组，则分组行也将进行计数，这将导致数据源中的项目的索引和Grid中的行索引不匹配。

解决这种问题的最简单的方式是枚举每一行，并将您搜索的项目和每一行的DataItem属性进行比较。下面的代码演示了如何做到这一点： C#

```
var customer = GetSomeCustomer;
#if false // ** 千万不要使用这种方式，因为它将在存在分组的情况下无法正常工作
int index = view.IndexOf(customer); if (index > -1)
{
_flex.Select(index, 0);
}
#else // 这是在Grid中搜索对象的安全方式
for (int row = 0; row <= _flex.Rows.Count; row++)
{
if (row.DataItem == customer)
{
_flex.Select(row, 0);
break;
}
```

```
}
#endif
```

自定义选择显示

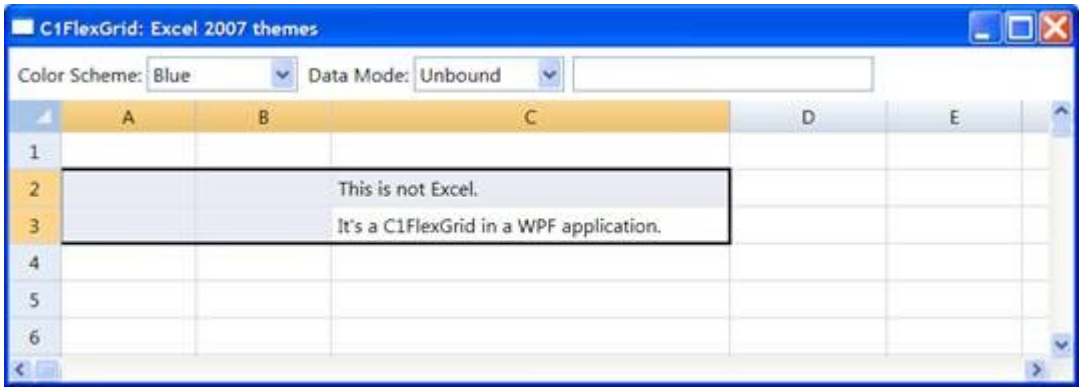
C1FlexGrid提供两种功能，允许您自定义选择高亮显示给最终用户的方式。

Excel-风格的Marquee

如果您设置了ShowMarquee属性的值为True，Grid将在选择区域自动绘制一个矩形框，使得非常容易的可以发现选择区域。默认情况下，Marquee是一个两个像素宽的黑色矩形框，但是您可以通过Marquee属性对其进行定制。

选中的单元格Header

如果您指定了自定义的画刷对象至Grid的ColumnHeaderSelectedBackground以及RowHeaderSelectedBackground属性，则Grid将高亮显示关联到选中单元格的Header，使得最终用户可以非常容易地确认包含选择区域的行和列的位置。
将这些属性综合在一起使用，使得您可以实现一个具有熟悉的Excel外观和使用习惯的Grid。下面的图像显示了这样的一个例子：



C1FlexGrid设计器具有一个允许您选择类似于Excel的预定义配色方案（Blue，Silver，Black）的上下文菜单。您可以容易地复制由设计器生成的XAML至可重复使用的样式资源。

编辑

自动完成并映射列

自动完成和映射列由一个内置的叫做ColumnValueConverter的类实现。这个类涉及三种常见的绑定场景：自动完成独占模式（列表框风格的编辑）
列只能取有限的特定值。例如，你有一个"Country"列，为字符串类型，同时具有一个国家名称的列表。用户应该从列表中选择一个国家，而不被允许输入的任何国家的名称不在此列表中。
您仅需两行代码即可处理这个场景：

C#

```
var
c = _flexEdit.Columns[
"Country"];
c.ValueConverter = new ColumnValueConverter(GetCountryNames(), true);
```

ColumnValueConverter构造器的第一个参数提供了一个合法值的列表。第二个参数确定是否最终用户允许输入在此列表中不存在的值（在此示例中，用户不允许这么做）。

自动完成非独占模式（组合框式编辑）

列具有一些共通的值选项，但是同样也可以接受其它值。例如，你有一个"Country"列，为字符串类型，并希望提供一个常见的国家名称列表供最终用户容易地进行选择。但在这种情况下，用户也应该允许键入在列表中不存在的值。
你仍然仅需要两行行代码即可处理这个场景： C#

```
var
```

```
c = _flexEdit.Columns[

"Country"];
c.ValueConverter = new ColumnValueConverter(GetCountryNames(), false);
```

和上一个示例一样，ColumnValueConverter构造器的第一个参数提供一组合法值的列表。在这个场景下，第二个参数决定了该列表是非排他性的，因此用户可以输入该列表中不存在的值。

使用一个键值字典自动完成

列将包含键值而不是实际的值。例如，该列可能包含一个整数，表示一个国家标识，但用户应该看到并编辑相应的国家名称。下面的代码演示了如何处理这个场景：

C#

```
// 建立键-值对应关系字典
var dct = new Dictionary<int, string>(); foreach (var country in GetCountryNames())
{

dct[dct.Count] = country; }


// 获取列
var

c = _flexEdit.Columns[

"CountryID"];
// 创建并为值字典分配Converter
c.ValueConverter = new ColumnValueConverter(dct);
// 将列靠左对齐
c.HorizontalAlignment = HorizontalAlignment.Left;
```

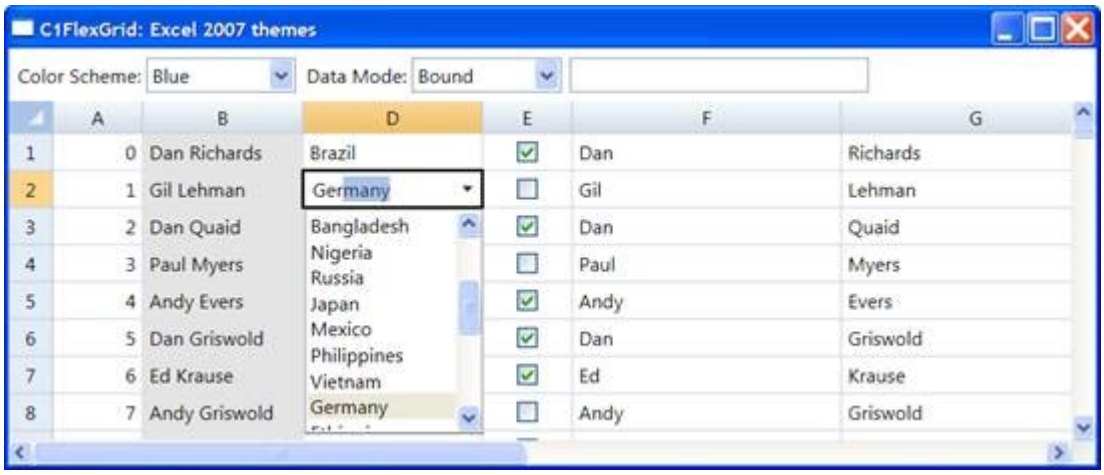
数据映射列

数据映射列包含键而不是实际值。例如，该列可能包含一个整数，表示一个国家标识，但用户应该看到并编辑相应的国家名称。
这种情况需要一点点的代码行：

C#

```
// 建立键-值字典
var dct = new Dictionary<int, string>(); foreach (var country in GetCountryNames())
{

dct[dct.Count] = country;


}
// 指定字典至列
var

c = _flexEdit.Columns[

"CountryID"];
c.ValueConverter = new ColumnValueConverter(dct);
c.HorizontalAlignment = HorizontalAlignment.Left;
```

代码首先是构建一个映射国家标识值（整数）到国家名称（字符串）的字典。然后使用词典构建ColumnValueConverter
并指定转换器至列的ValueConverter属性，就像之前的示例那样。用户可以选择在字典里现有的国家，且不允许输入任何未经映射的值。最后，该代码将该列的对齐方式设置为左对齐。由于该列实际上包含整数值，它将默认为右对齐。但是，因为我们现在需要显示国家的名字，在这里左对齐是一个更好的选择。
下图显示编辑器的外观，同时从列表中选择一个值。注意这里编辑器是如何支持自动完成的，当用户键入"Ger"下拉框将自动选择有效的选项"Germany"（而不是"Guatemala"，或者"Eritrea"，再或者"Romania"）。



使用自定义编辑器

C1FlexGrid提供两种内置的编辑器：一个复选框用于编辑布尔类型的值，以及一个C1FlexComboBox，用于使用上面所述的自动完成功能扩展常规的TextBox。
您可以使用和该文档中之前描述的创建自定义单元格相同的机制创建并使用您自己的编辑器。实现一个自定义的CellFactory
类并重写CreateCellEditor 方法创建并绑定您的编辑器至底层数据的值。使用XAML为需要自定义编辑器的列指定CellEditingTemplate。

配置编辑器

无论您使用的是内置的或自定义的编辑器，你可以利用PrepareCellForEdit
事件在编辑器激活之前配置该编辑器。例如，下面的代码将更改编辑器的选择状态为蓝色底色，以黄色显示：

C#

```
// 挂载事件处理函数
_grid.PrepareCellForEdit += _grid_PrepareCellForEdit;
// 通过修改选择的外观自定义编辑器
void _grid_PrepareCellForEdit(object sender, CellEditEventArgs e)
{
var b = e.Editor as Border; var tb = b.Child as TextBox;
tb.SelectionBackground = new SolidColorBrush(Colors.Blue); tb.SelectionForeground = new SolidColorBrush(Colors.Yellow); }
```

分组

分组数据（使用ICollectionView）

ICollectionView接口包含对分组的支持，这将允许您创建分层的数据视图。例如，在上面的示例中，如果要对客户按照不同的国家和城市进行分组，您只需要简单地将生成Grid数据的部分修改为以下：

C#

```
List<Customer> list = GetCustomerList();
PagedCollectionView view = new PagedCollectionView(list); using (view.DeferRefresh())
{
view.GroupDescriptions.Clear();
view.GroupDescriptions.Add(new PropertyGroupDescription("Country")); view.GroupDescriptions.Add(new PropertyGroupDescription
("Active")); } _flexGrid.ItemsSource = view;
```

表达式 using(view.DeferRefresh())
为可选。它通过挂起来自于数据源的通知直至全部的分组设置完毕，以提高性能。下图显示执行的结果：

数据项按照不同的国家以及其活动的区域进行分组。用户可以单击分组Header上的图标以折叠或者展开分组，就像他们在操作一个TreeView控件一样。
如果您希望在Grid层面禁用分组，请设置Grid的GroupRowPosition属性的值为GroupRowPosition.None（其他可用的选项为AboveData以及BelowData）。
由ICollectionView类提供的分组机制简单但功能强大。每一个级别的分组由PropertyGroupDescription对象定义。该对象允许您选择用于进行分组的属性，以及一个ValueConverter用来决定如何在分组时使用这个属性。
例如，如果我们想按国家的首字母来分组，则可以简单地修改代码如下：

C#

```
List<Customer> list = GetCustomerList();
PagedCollectionView view = new PagedCollectionView(list); using (view.DeferRefresh())
{
view.GroupDescriptions.Clear();
view.GroupDescriptions.Add(new PropertyGroupDescription("Country")); view.GroupDescriptions.Add(new PropertyGroupDescription
("Active")); var

gd = view.GroupDescriptions[0]

as PropertyGroupDescription; gd.Converter = new CountryInitialConverter();
}
_flexGrid.ItemsSource = view;
```

CountryInitialConverter类实现了IValueConverter接口。它返回国家名称的第一个英文字母，使用该字母进行分组而不再是使用完整的国家名称。

C#

```
// 按照首字母对国家进行分组的转换器
class CountryInitialConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return ((string

)value)[0].ToString().ToUpper();


}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
throw new NotImplementedException();
}
}
```

在完成这个小小的变化之后，客户将按照所在国家的第一个字母而不再是按照国家名称进行分组：

注意，该分组行将显示该分组的信息（依照分组的属性以及其值，以及其中包含的项目个数）。为定制此信息，可以创建一个新的IValueConverter 类并将其指定给Grid的GroupHeaderConverter 属性。
例如，默认的分组Header转换器（显示如图片所示信息的那个默认转换器）的实现如下所示：

C#

```
// 用作格式化显示分组标题的类
public class GroupHeaderConverter : IValueConverter
{
public object Convert(object value,
Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
var gr = parameter as GroupRow; var group = gr.Group;
if (group != null && gr != null && targetType == typeof(string))
{
var

desc = gr.Grid.View.GroupDescriptions[gr.Level]

as
PropertyGroupDescription; return desc != null
? string.Format("{0}: {1} ({2:n0} items)",
desc.PropertyName, group.Name, group.ItemCount)
: string.Format("{0} ({1:n0} items)",
group.Name, group.ItemCount);
}
return value;
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return value;
}
}
```

通过C1FlexGridGroupPanel管理分组

C1FlexGridGroupPanel是一个新控件，提供了在C1FlexGrid控件中管理分组的UI。
C1FlexGridGroupPanel控件目前在Silverlight以及WPF版本中可用。它实现在一个单独的程序集中，这取决于平台。
C1.WPF.FlexGrid.GroupPanel.4.dll
C1.Silverlight.FlexGrid.GroupPanel.5.dll

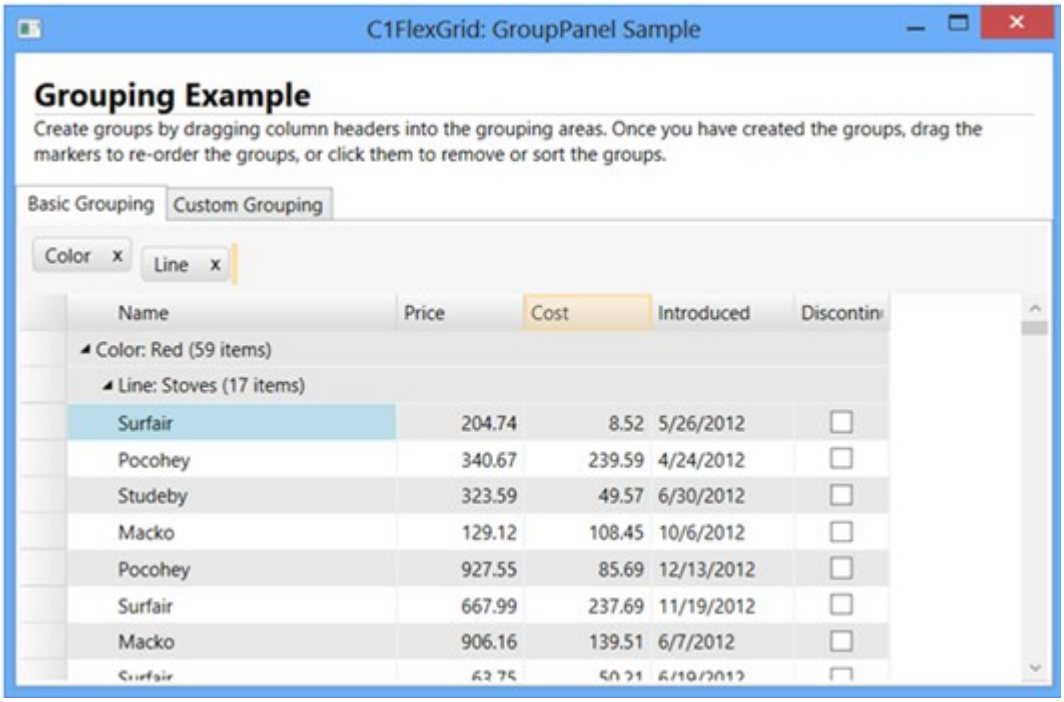
使用C1FlexGridGroupPanel控件

为使用C1FlexGridGroupPanel控件，添加该控件在窗体上的某个C1FlexGrid控件上方，并设置其FlexGrid属性，使得其引用该C1FlexGrid用来显示数据。例如：

XAML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition />
</Grid.RowDefinitions>
<c1:C1FlexGridGroupPanel
Background="WhiteSmoke"
FlexGrid="{Binding ElementName=_flex}"/>
<c1:C1FlexGrid x:Name="_flex" Grid.Row="1" /> </Grid>
```

该段XAML是非常简单的。C1FlexGridGroupPanel不需要设置样式，因为它将从控制的C1FlexGrid获取所需的属性设置。如果您改变了列Header的背景，前景或者字体，则C1FlexGridGroupPanel
将自动地使用这些元素来渲染分组标记，使得其看起来和列Header一样。该段XAML将产生下图所示结果：

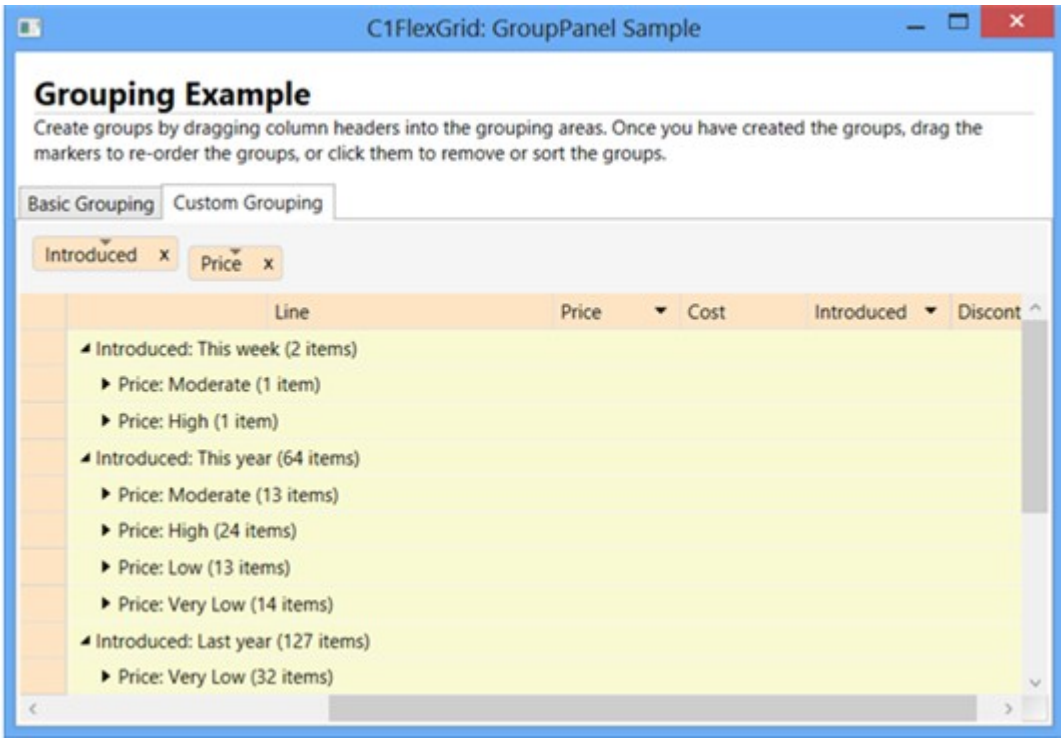


该图像显示了按照"Color"以及"Line"进行分组的产品的C1FlexGrid。该分组通过拖拽Grid列至分组区操作进行创建。
在本图像中，"Cost"列高亮显示，因为它正在被拖到分组区域。位于"Line"分组标记右侧的橙色光标指示新分组插入的位置。
您可以通过设置DragMarkerColor属性改变拖拽标记的颜色（图中所示为橙色）。一旦一个分组被创建，则关联的列默认将被隐藏。这是一个可选的功能，您可以通过设置HideGroupedColumns 属性的值为false关闭该设置。
您可以单击分组标记以便按照升序或者降序的方式对分组中的数据进行排序。按下Control键并单击分组Header以移除排序状态。这是和在其他平台下的C1FlexGrid相同的行为。
您也可以在分组区域拖拽分组标记以便重新排布分组，或者将其拖拽回Grid区域已撤销分组，并将该列恢复到任意位置。分组标记也具有关闭按钮（"×"），您可以单击以删除该分组。
C1FlexGridGroupPanel通过用作Grid数据源的ICollectionView接口管理分组。对分组所做的全部改变对绑定到相同的ICollectionView的控件可见。注意，这意味着分组特性不能用于非绑定使用场景。
当C1FlexGridGroupPanel创建了一个新的分组时，它将触发一个PropertyGroupCreated事件，允许应用程序自定义该新建的分组。该事件通常用来为新创建的分组指定自定义转换器。例如：

C#

```
/// <summary>
/// 自定义由C1FlexGridGroupPanel创建的分组描述。
/// </summary>
void _groupPanel_PropertyGroupCreated(object sender, PropertyGroupCreatedEventArgs e) {
var pgd = e.PropertyGroupDescription; switch (pgd.PropertyName)
{ case "Introduced":
pgd.Converter = new DateTimeGroupConverter(); break; case "Price":
pgd.Converter = new AmountGroupConverter(1000); break; case "Cost":
pgd.Converter = new AmountGroupConverter(300); break;
}
}
```

该代码处理了PropertyGroupCreated事件，并指定自定义转换器至数据源的不同列。在本示例中，
DateTimeGroupConverter以及AmountGroupConverter类是用来对DateTime以及双精度类型的值转换为不同范围的简单转换器。
下图显示了自定义分组的效果：



请注意项目可能出现在多个不同的分组中。例如，DateTimeGroupConverter将日期分组为"本周"，"本年度"，"去年"，以及"去年以前"。"本周"分组的项目也包括在"本年度"分组中。
这是ICollectionView接口本身提供的功能，并不是C1FlexGrid或者C1FlexGridGroupPanel提供的特定功能。

C1FlexGridGroupPanel实现注意事项

我们决定将分组UI的功能提供在一个独立的程序集而不是将其直接添加在C1FlexGrid控件上，原因如下：
尽可能地保持C1FlexGrid轻巧，快速，可移植。Silverlight版本的C1FlexGrid仍旧小于250K，这使得其始终保持目前市售版本的data grid类产品中最小的尺寸。它也是最广泛使用的data grid，包括WinForms，
Silverlight，WPF，Windows Phone，WinRT，ActiveX，以及Compact Framework版本。
允许轻松定制的分组界面。C1FlexGridGroupPanel是一个非常简单的控件，我们将源代码提供给用户，使得用户可以创建其自定义的版本。
为演示C1FlexGrid控件的可扩展性和灵活性。C1FlexGridGroupPanel使用由C1FlexGrid暴露的简单强大的对象模型进行开发，不使用任何的自定义扩展，修改，或者使用任何内部的后门方法。（与此相同的还有
C1FlexGridFilter组件，该组件提供了自组织的过滤，并且同样地包含在另一个独立的程序集中）。
C1FlexGridGroupPanel实际上是一个Grid元素，包含一个用来显示水印消息的TextBlock，以及一个用来显示源
ICollectionView中所有可用分组的水平放置的StackPanel。
分组由GroupMarker元素表示，该元素可以被单击对分组进行排序或者收起整个分组，再或者可以进行拖拽以重新排布分组的顺序。
C1FlexGridGroupPanel处理四种类型的拖拽行为：
在分组区域内拖拽GroupMarkers以便重新排布这些分组，拖拽GroupMarkers至Grid区域，以便移除该分组并在特定位置恢复该列，从Grid中拖拽ColumnHeader元素至分组区域以创建新的分组，以及在Grid中间拖拽ColumnHeader元素以便重新排布Grid的列。
所有的拖拽行为由一个叫做DragDropManager的辅助类处理。
前两种拖拽行为由GroupMarker类发起，将检测鼠标拖拽动作并调用DragDropManager上的DoDragDrop，同时传入
Marker做为一个参数。后两种拖拽行为将在响应C1FlexGrid的DraggingColumn事件的事件处理函数中发起。
当DoDragDrop方法被调用时，DragDropManager将在整个页面显示一个透明的元素，捕获鼠标消息，并触发
Dragging事件，使得调用方可以更新拖放的目标位置。当用户释放鼠标时，DragDropManager将触发Dropped事件，调用方可以结束整个拖放动作。
该实现非常简短并且具有可移植性。程序集仅有25K左右，并且代码可以在Silverlight或者WPF下进行编译，完全不需要任何的条件编译代码块。 6.3.5 聚合数据
一旦您对数据进行了分组，则通常下一步有用的操作时计算这些分组的聚合值。例如，如果您将销售数据按国家或产品类别分组，您可以显示每个国家和产品类别的总销售额。为了使用C1FlexGrid实现这一点，可以设置列上的
GroupAggregate属性为聚合，则Grid将自动地计算并显示聚合结果。当数据发生变化时，聚合结果将自动地重新计算。请注意，聚合将出现在该分组的Header行中。设置Grid的AreGroupHeadersFrozen属性的值为False以使得其可见。
例如，考虑以下的Grid定义：

XAML

```
<fg:C1FlexGrid x:Name="_flex" AutoGenerateColumns="False">
<fg:C1FlexGrid.Columns>
<fg:Column Header="Line" Binding="{Binding Line}" />
<fg:Column Header="Color" Binding="{Binding Color}" />
<fg:Column Header="Name" Binding="{Binding Name}" />
<fg:Column Header="Price" Binding="{Binding Price}"
Format="n2" HorizontalAlignment="Right" Width="*"/> <fg:Column Header="Cost" Binding="{Binding Cost}"
Format="n2" HorizontalAlignment="Right" Width="*"/> <fg:Column Header="Weight" Binding="{Binding Weight}" Format="n2" Horizo
```

```
ntalAlignment="Right" Width="*"/> <fg:Column Header="Volume" Binding="{Binding Volume}" Format="n2" HorizontalAlignment="Rig
ht" Width="*"/>
</fg:C1FlexGrid.Columns> </fg:C1FlexGrid>
```

如果您设置了Grid的ItemsSource为一个ICollectionView对象，该对象设置为按照"Line"，"Color"以及"Name"进行分组，则您将获得一个类似于下图的Grid：

Line	Color	Name	Price	Cost	Weight	Volume
▲ Total: (200 items)						
▲ Line: Washers (55 items)						
▲ Color: Green (16 items)						
▲ Price: Medium (1 items)						
Washers	Green	P 0	68.00	233.00	3.00	1,171.00
▲ Price: Very High (8 items)						
Washers	Green	P 2	678.00	201.00	12.00	2,748.00
Washers	Green	P 23	840.00	283.00	60.00	2,077.00
Washers	Green	P 42	687.00	107.00	59.00	1,856.00
Washers	Green	P 88	747.00	408.00	88.00	899.00
Washers	Green	P 91	630.00	249.00	82.00	3,505.00
Washers	Green	P 132	658.00	15.00	13.00	3,857.00
Washers	Green	P 187	889.00	349.00	68.00	2,952.00
Washers	Green	P 194	865.00	185.00	20.00	582.00
▲ Price: High (6 items)						
Washers	Green	P 67	487.00	521.00	65.00	2,196.00
Washers	Green	P 75	257.00	350.00	19.00	526.00

该Grid以一个可折叠的大纲试图格式显示这些分组，并自动地显示在每一个分组中元素的个数。这一点和Microsoft DataGrid控件显示的视图非常相似。

C1FlexGrid让您更进一步，以显示列的聚合值。例如，为了显示"Price"，"Cost"，"Weight"以及"Volume"列的合计，请按照如下方式修改XAML：

XAML

```
<fg:C1FlexGrid x:Name="_flex" AutoGenerateColumns="False"
AreRowGroupHeadersFrozen="False">
<fg:C1FlexGrid.Columns>
<fg:Column Header="Line" Binding="{Binding Line}" />
<fg:Column Header="Color" Binding="{Binding Color}" />
<fg:Column Header="Name" Binding="{Binding Name}" />
<fg:Column Header="Price" Binding="{Binding Price}" Format="n2" HorizontalAlignment="Right" Width="*"
GroupAggregate="Sum"/>
<fg:Column Header="Cost" Binding="{Binding Cost}"
Format="n2" HorizontalAlignment="Right" Width="*"
GroupAggregate="Sum"/>
<fg:Column Header="Weight" Binding="{Binding Weight}"
Format="n2" HorizontalAlignment="Right" Width="*"
GroupAggregate="Sum"/>
<fg:Column Header="Volume" Binding="{Binding Volume}"
Format="n2" HorizontalAlignment="Right" Width="*"
GroupAggregate="Sum"/>
</fg:C1FlexGrid.Columns> </fg:C1FlexGrid>
```

该段XAML代码包含两点变化：

设置Grid的AreGroupHeadersFrozen属性的值为False。这个设置是必要的，因为分组的聚合信息将出现在分组的Header行上，如果Header被冻结，则聚合信息将不可见。

同时还设置了若干列的GroupAggregate属性的值为"Sum"。这将使得Grid计算并在分组的Header行上显示聚合信息。有这么几个聚合选项可用，包括"Sum"，"Average"，"Count"，"Minimum"，"Maximum"等等。

做了这个改变后，Grid将看起来像这样：

Line	Color	Name	Price	Cost	Weight	Volume
◢ Total: (200 items)			103,610.00	61,744.00	10,089.00	573,086.00
◢ Line: Washers (55 items)			27,656.00	16,865.00	2,697.00	144,544.00
◢ Color: Green (16 items)			8,150.00	4,658.00	669.00	33,853.00
◢ Price: Medium (1 items)			68.00	233.00	3.00	1,171.00
Washers	Green	P 0	68.00	233.00	3.00	1,171.00
◢ Price: Very High (8 items)			5,994.00	1,797.00	402.00	18,476.00
Washers	Green	P 2	678.00	201.00	12.00	2,748.00
Washers	Green	P 23	840.00	283.00	60.00	2,077.00
Washers	Green	P 42	687.00	107.00	59.00	1,856.00
Washers	Green	P 88	747.00	408.00	88.00	899.00
Washers	Green	P 91	630.00	249.00	82.00	3,505.00
Washers	Green	P 132	658.00	15.00	13.00	3,857.00
Washers	Green	P 187	889.00	349.00	68.00	2,952.00
Washers	Green	P 194	865.00	185.00	20.00	582.00
◢ Price: High (6 items)			2,084.00	2,476.00	193.00	12,436.00
Washers	Green	P 67	487.00	521.00	65.00	2,196.00
Washers	Green	P 75	257.00	350.00	19.00	526.00

请注意，分组Header是如何显示分组的聚合值的。当数据发生变化时，聚合值将自动地重新计算。

合并

合并单元格

Grid上的AllowMerging属性将在Grid级别启用单元格合并。一旦您在Grid级别启用了合并，则可以使用
Row.AllowMerging以及Column.AllowMerging属性以选择特定的行和列进行合并。例如，下面的代码合并包含相同国家的单元格：

C#

```
// 在可滚动区域启用合并
fg.AllowMerging = AllowMerging.Cells;
// 在列"Country" 以及"FirstName"上启用合并

fg.Columns[

"Country"].AllowMerging = true

; fg.Columns[

"FirstName"].AllowMerging = true;
```

此代码将创建如下图所示grid：

Country	Active	ID	Name	First
◢ Country: M (39 items)				
◢ Active: False (15 items)				
Mexico	☐	9	Ben Myers	Ben
	☐	110	Steve Orsted	Steve
	☐	246	Zeb Myers	Zeb
	☐	257	Steve Heath	Steve
	☐	261	Mark Stevens	Mark
	☐	410	Noah Quaid	Noah
	☐	448	Karl Stevens	Karl
	☐	490	Larry Stevens	Larry
Myanmar	☐	80	Fred Ulam	Fred
	☐	262	Zeb Frommer	Zeb
	☐	314	Karl Ulam	Karl
	☐	324	Ulrich Orsted	Ulrich
	☐	341	Oprah Griswold	Oprah
	☐	427	Noah Neiman	Noah

过滤

C1FlexGrid和"扩展程序集"一起打包发布，这些扩展程序集包括C1.Silverlight.FlexGridFilter 和
C1.WPF.FlexGridFilter（在线文档'C1.WPF.FlexGridFilter.4
程序集'），这些程序集提供了Excel样式的过滤功能。为使用这些组件，需要将它们添加到您的工程引用，并创建一个C1FlexGridFilter
对象关联到一个现有的Grid。例如：

C#

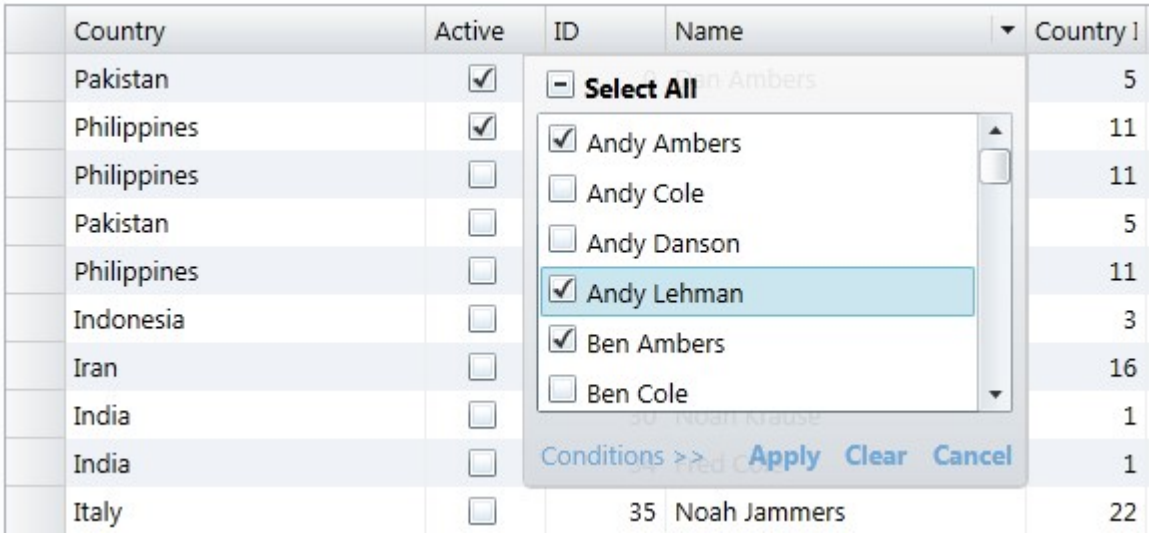
```
// 创建C1FlexGrid var flex = new C1FlexGrid();
// 在Grid上启用过滤
var gridFilter = new C1FlexGridFilter(flex);
```

此外，C1.Silverlight.FlexGrid.GroupPanel以及C1.WPF.FlexGrid.GroupPanel （在线文档'C1.WPF.FlexGrid.GroupPanel.4
程序集'）程序集同样也做为扩展程序集提供对C1FlexGrid控件的分组管理功能。
我们决定使用扩展组件而不是将功能直接添加到控件上的原因有两个：能够使我们保持Grid所在的程序集为一个较小的尺寸，并允许您选择在
每一个工程中使用哪些扩展。
允许通过自定义代码扩展C1FlexGrid的功能。该代码示例演示了由WPF及Silverlight版提供的ExcelBook以及ExcelGrid的功能。
您同样也可以在Grid声明的XAML文件中启用过滤。这里是实现该功能的语法：

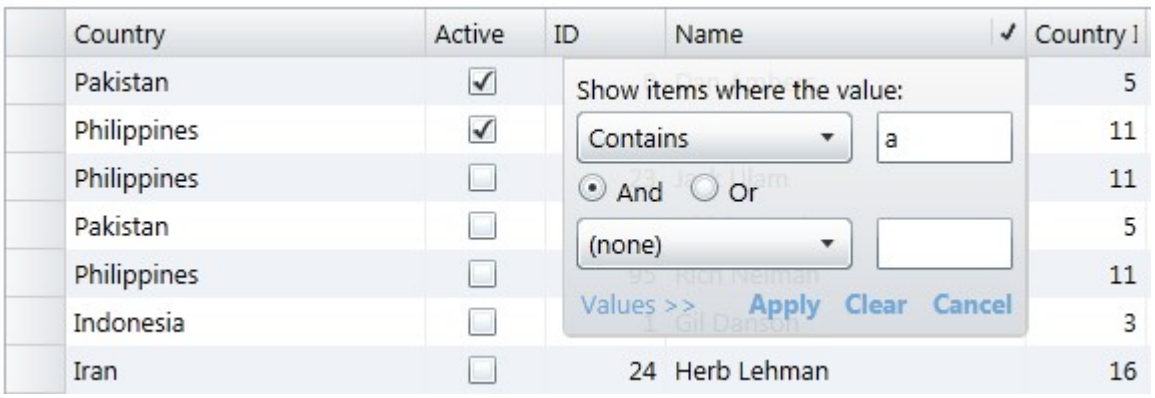
XAML

```
<c1:C1FlexGrid Name="_flex" >
<!-- 向控件添加过滤支持： -->
<c1:C1FlexGridFilterService.FlexGridFilter>
<c1:C1FlexGridFilter />
</c1:C1FlexGridFilterService.FlexGridFilter> </c1:C1FlexGrid>
```

一旦启用了过滤功能，当鼠标悬停在列头上时，Grid会显示一个下拉图标。下拉菜单中显示一个编辑器，允许用户指定如何筛选该列中的数据
。用户可选择两种类型的过滤器：值过滤器值过滤器：该过滤器允许用户从列表中显示的值中进行选择。
条件过滤器条件过滤器：此过滤器允许用户指定由一个操作符（大于，小于等等）和一个参数。条件本身可以通过一个
AND或者一个OR运算符进行组合。
下面的图像显示，当正在进行编辑时，过滤器的外观：



值过滤器值过滤器：用户通过从列表中选择值创建筛选器。



条件过滤器条件过滤器：用户通过设置一个或两个条件创建过滤器。
对于大多数应用程序，默认的过滤器设置是足够的，但您可以通过几种不同的方法自定义筛选器。

选择筛选模式

过滤器工作在两种不同的模式下，取决于UseCollectionView属性的设置。
如果您设置UseCollectionView属性的值为false，则不满足过滤条件的行将隐藏（过滤器将设置它们的Visible属性为false）。在这种模式下，过滤器对行计数没有影响。您可以在绑定以及非绑定模式的Grid中使用该模式。
如果您设置了过滤器的UseCollectionView属性值为true，则过滤将应用到数据源（通过ICollectionView.Filter属性）。在这种模式下，对过滤器所做的改变将直接影响数据源暴露给Grid的数据项的个数，与此同时，任何绑定到相同数据源的其他控件将受到影响。您只能在绑定模式使用该模式的过滤器。
例如：

C#

```
// 创建C1FlexGrid
var flex = new C1FlexGrid();
// 在Grid上启用过滤
var gridFilter = new C1FlexGridFilter(flex);
// 在数据源级别进行过滤
gridFilter.UseCollectionView = true;
```

或者，通过 XAML:

XAML

```
<c1:C1FlexGrid Name="_flex" >
<!-- 向控件添加过滤支持： -->
<c1:C1FlexGridFilterService.FlexGridFilter>
<c1:C1FlexGridFilter UseCollectionView="True"/>
</c1:C1FlexGridFilterService.FlexGridFilter> </c1:C1FlexGrid>
```

为每一列自定义过滤器类型

默认情况下，每个列将启用过滤器。包含布尔型或者枚举值的数据列将使用一个值过滤器，而包含其他数据值类型的列将可以使用值过滤器或者条件过滤器。您可以通过FilterType属性改变这一行为并指定在每一列上启用的过滤器类型。在当列举有大量的相同值或者当列包含无法进行过滤的绑定关系时，指定过滤器类型将非常重要。
例如，包含图像的列不能用值或条件筛选器来过滤。在这种情况下，你可以通过设置FilterType属性为None禁用过滤器。一个包含数千个项目的Grid可能具有一个唯一标识符的列，该列添加了过多的项目至值过滤器，这将使得其性能低下，实际上这种过滤通常没有什么实际用途。在这种情况下，通过设置FilterType属性为Condition以禁用值过滤器。
下面的代码演示了如何实现这一点：

C#

```
// 创建C1FlexGrid var flex = new C1FlexGrid();
// 在Grid上启用过滤
var gridFilter = new C1FlexGridFilter(flex);
// 在图像类型的列上禁用过滤
var

columnFilter = gridFilter.GetColumnFilter(flex.Columns["Image"]); columnFilter.FilterType = FilterType.None;


// 在标识符列上禁用值过滤器

columnFilter = gridFilter.GetColumnFilter(flex.Columns["ID"]); columnFilter.FilterType = FilterType.Condition;
```

在代码中指定过滤器

在大多数情况下，用户仅需设置过滤器。但是ColumnFilter类同样也提供了完整的对象模型，允许开发人员通过代码检查以及修改过滤条件。
例如，下面的代码将过滤器应用到第二列。该过滤器将使的Grid显示第二列中的值包含字母Z的项目：

C#

```
// 创建C1FlexGrid var flex = new C1FlexGrid();
// 在Grid上启用过滤
var gridFilter = new C1FlexGridFilter(flex);
// 获取第一列的过滤器
var

columnFilter = gridFilter.GetColumnFilter(flex.Columns[0]);


// 创建过滤条件（包含字母'Z'）
var condition = columnFilter.ConditionFilter.Condition1; condition.Operator = ConditionOperator.Contains;
condition.Parameter = "Z";
// 应用该过滤器
gridFilter.Apply();
```

保存过滤器

C1FlexGridFilter类包含一个FilterDefinition属性，该属性可以以XML字符串形式获取或设置当前的过滤器状态。当用户退出应用程序时，可以使用该字符串来保持该过滤器状态，这样您可以稍后还原。
您也可以保存几个过滤器的定义，允许用户选择并自定义这些预设过滤器。您还可以通过SaveFilterDefinition 以及LoadFilterDefinition方法保存和恢复过滤器的定义。

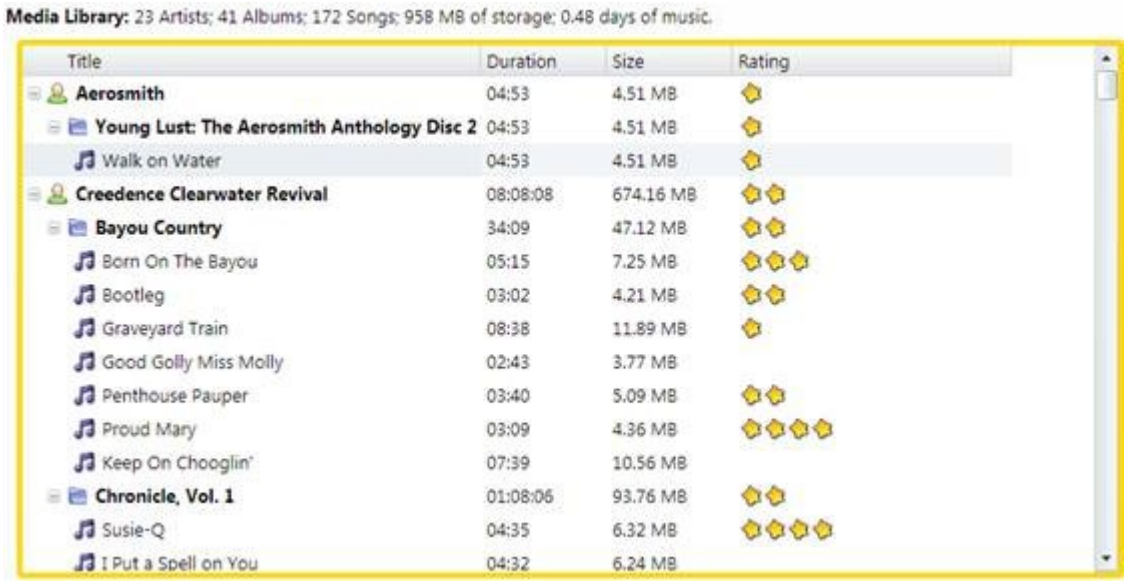
6.5.2 过滤数据（使用ICollectionView）

ICollectionView接口通过其Filter属性支持数据过滤。Filter属性指定一个方法，该方法将为集合中的每一个数据项进行调用。如果方法返回true，则该项目包含在视图中。如果方法返回false，则该项目将被过滤掉。（这种方法被称为断言）。
该文档包含的MainTestApplication示例包括一个SearchBox控件，该控件包含一个文本框，在此用户可以键入一个待搜索的值，除此之外，还包含了一个Timer。定时器提供了一个小的延迟，允许用户在输入待搜索值时，不会再每一个字符键入时立刻重复应用搜索。
当用户停止输入，定时器经过一段时间后将通过以下代码应用过滤器：

C#

```
_view.Filter = null;
_view.Filter = (object item) =>
{ var srch = _txtSearch.Text; if (string.IsNullOrEmpty(srch))
{ return true;
} foreach (PropertyInfo pi in _propertyInfo)
{ var value = pi.GetValue(item, null) as string; if (value != null && value.IndexOf(srch,
StringComparison.OrdinalIgnoreCase) > -1)
{ return true;
}
} return false;
};
```

注意这里代码是如何使用一个lambda函数设置Filter属性的。我们可以提供一个单独的方法，但这种符号通常更方便，因为它是简洁的，并且允许我们使用本地变量，如果我们需要它们。
lambda函数接受一个项目作为一个参数，获取该对象的指定属性的值，如果任何对象的属性包含待搜索的字符串则返回true。
例如，如果对象是"歌曲"类型，并且指定的属性是"标题"，"专辑"，"艺术家"，如果在歌曲的标题，专辑或艺术家中包含待搜索的字符串则返回true。这是一个功能强大且易于使用的搜索机制，类似于一个用于在苹果的iTunes应用程序。
一旦应用了过滤器，grid（以及其他绑定到相同的ICollectionView对象的控件）都将仅显示由该过滤器筛选出来的项目，以反映该变化。
注意过滤和分组可以一起工作。下图（来自于MainTestApplication 示例）显示一个非常大的歌曲列表，并应用了一个过滤器：



The image shows the filter set to the word "Water." The filter looks for matches in all fields (song, album, artist), so all "Creedence Clearwater Revival" songs are automatically included.
Notice the status label above the grid. It automatically updates whenever the list changes, so when the filter is applied the status updates to reflect the new filter. The routine that updates the status uses LINQ to calculate the number of artists, albums, and songs selected, as well as the total storage and play time. The song status update routine is implemented as follows:

C#

```
// update song status void UpdateSongStatus()
{
var view = _flexiTunes.ItemsSource as ICollectionView; var songs = view.OfType<Song>();
_txtSongs.Text = string.Format(
"{0:n0} Artists; {1:n0} Albums; {2:n0} Songs; " +
"{3:n0} MB of storage; {4:n2} days of music.",
(from s in songs select s.Artist).Distinct().Count(),
(from s in songs select s.Album).Distinct().Count(),
(from s in songs select s.Name).Count(),
(double)(from s in songs select s.Size/1024.0/1024.0).Sum(),
(double)(from s in songs select s.Duration/3600000.0/24.0).Sum()); }
```

This routine is not directly related to the grid, but is listed here because it shows how you can leverage the power of LINQ to summarize status information that is often necessary when showing grids bound to large data sources.
The LINQ statement above uses the Distinct and Count commands to calculate the number of artists, albums, and songs currently exposed by the data source. It also uses the Sum command to calculate the total storage and play time for the current selection.