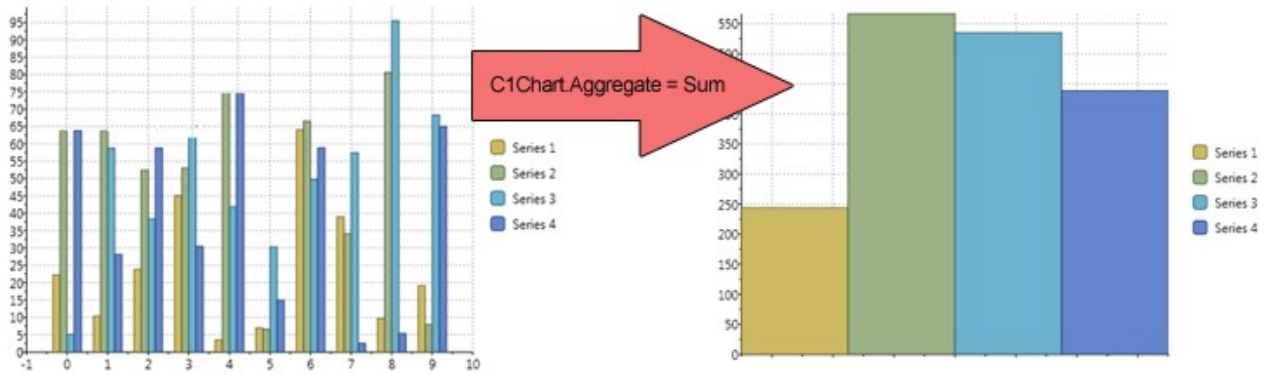
DataSeries 聚合

C1Chart控件可以自动地将每一个数据系列分组并聚合为一个单独的绘图值，仅需设置一个属性即可完成。这个动作将一个单一系列的所有值组合成一个值，使用一个支持的聚合函数（总和、平均、最小、最大、方差、标准偏差等）。
仅需要设置C1Chart控件上的Aggregate属性即可对全部的数据系列应用分组。例如，如果我们有一个图表，有四个数据序列，我们设置Aggregate属性的值为Sum，则结果如下图所示：



用于对DataSeries进行聚合的标记语法类似以下：

XAML

```
<c1chart:C1Chart x:Name="chart0" Height="350" Width="450" ChartType="Column"

Palette="Solstice" Foreground="Black" >
<!-- Populate the chart with three series -->
<c1chart:C1Chart.Data>
<c1chart:ChartData >
<c1chart:ChartData.Children>
<c1chart:DataSeries Label="Revenue" Aggregate="Sum"
Values="1200, 1205, 400, 1410" ></c1chart:DataSeries>
<c1chart:DataSeries Label="Expense" Aggregate="Sum"
Values="400, 300, 300, 210" ></c1chart:DataSeries>
<c1chart:DataSeries Label="Profit" Aggregate="Sum"
Values="790, 990, 175, 1205" ></c1chart:DataSeries>
</c1chart:ChartData.Children>
</c1chart:ChartData>
</c1chart:C1Chart.Data>
</c1chart >
```

日期时间分组

C1Chart（在线文档 'C1Chart 类'）控件在任何数据系列上都允许自定义聚合。通过为AggregateGroupSelector（在线文档 'AggregateGroupSelector
委托'）属性定义您自己自定义聚合逻辑，C1Chart控件可以按照任何您期望的方式对数据进行分组。例如，您可以在日期字段中提供分组，以汇总每月或每年的值。你甚至可以建立自己的数值范围和类别对数据点进行分组。
本主题假定您在XAML中创建了一个C1Chart
控制并把它命名为"c1chart1"。关于如何通过XAML创建一个控件的更多信息，请参见快速入门（在线文档）或者概念和主要属性主题。
为了创建自定义聚合函数，首先需要创建待汇总的数据。您可以创建一个简单的业务对象，具有两个属性：Value（双精度浮点数）和日期（日期类型）。在下面的例子中，这个业务对象被称为SampleItem。做为参考，您可以在本主题的底部找到这个类型的定义。创建一个包含随机数据的ObservableCollection：

C#

```
Random rnd = new Random();
```

```
ObservableCollection<SampleItem> _items = new ObservableCollection<SampleItem>(); for(int i = 0; i < 400; i++)
{
_items.Add(new SampleItem { Value = rnd.Next(0, 100), Date =
DateTime.Now.AddDays }); }
```

之后绑定您的XYDataSeries至项目的集合：

C#

```
// 配置数据系列
var ds = new XYDataSeries()
{
ItemsSource = _items,
ValueBinding = new Binding { Path = new PropertyPath("Value") },

XValueBinding = new Binding { Path = new PropertyPath("Date") },

Aggregate = Aggregate.Sum,
AggregateGroupSelector = GroupSelectorByDate,
Label = "Sales"
};
```

您在上面的代码中设置了两个关键属性：Aggregate 和AggregateGroupSelector。Aggregate 属性决定用作聚合图表数据的函数。AggregateGroupSelector
属性决定为数据系列提供分组选择器的函数。在您设置了自定义函数之前，虽说添加系列到图表可以设置沿着x-轴显示日期。您还可以设置X轴为显示时间，因此日期显示正确：

C#

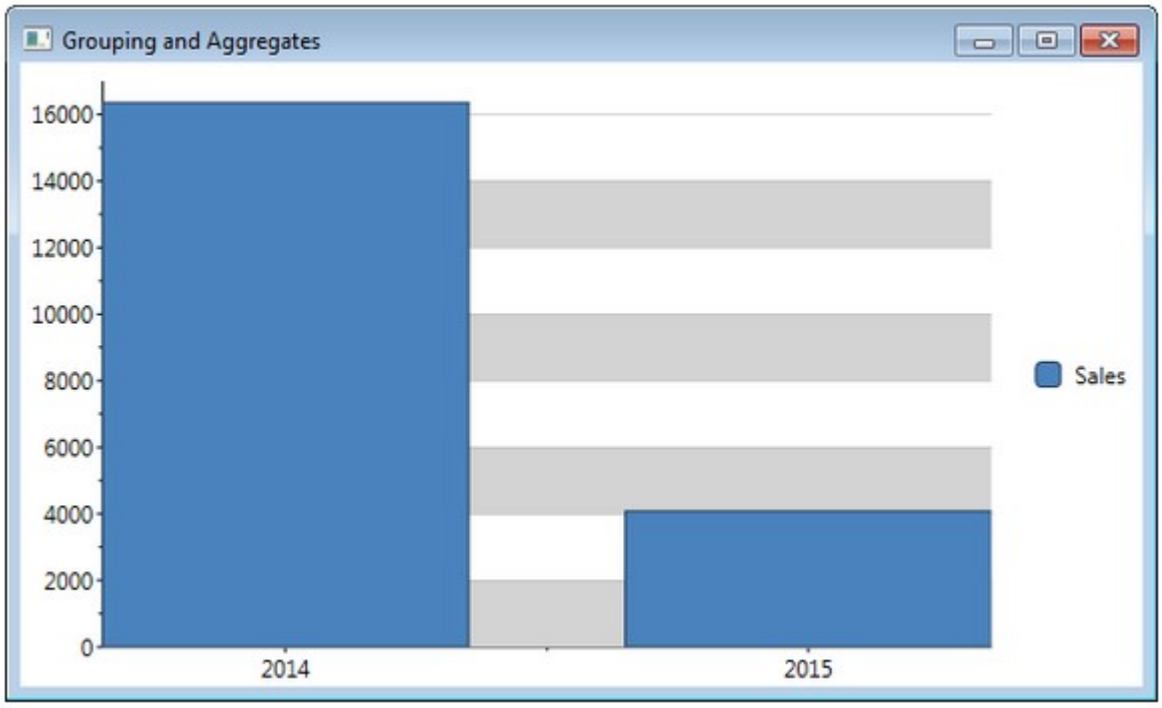
```
// 配置图表
c1Chart1.BeginUpdate();
c1Chart1.ChartType = ChartType.Column;
// 添加数据系列
c1Chart1.Data.Children.Add(ds);
// 使用特定格式的时间坐标轴
c1Chart1.View.AxisX.IsTime = true; c1Chart1.View.AxisX.AnnoFormat = "yyyy"; c1Chart1.View.AxisX.UseExactLimits = true;
// 应用一些风格
c1Chart1.View.AxisX.MajorGridStrokeThickness = 0; c1Chart1.View.AxisY.MajorGridFill = new SolidColorBrush(Colors.LightGray
); c1Chart1.EndUpdate();
```

注意，当您需要沿x轴显示日期，您需要设置的时间属性设置为true。您将按年进行分组，因此请注意，AnnoFormat属性已经被设置为显示完整的年份。下面的代码定义了GroupSelectorByDate 函数。该函数将调用您的图表中的每个数据点，并确定该数据所属的组。

C#

```
double GroupSelectorByDate(double x, double y, object o)

{

// 将年份作为双精度浮点数值返回
DateTime dt = x.FromOADate();
// 为了按照年份进行分组，我们返回日期的年份信息
// 同时也设置 AnnoFormat 为 "yyyy"
return new DateTime(dt.Year, 1, 1).ToOADate();
}
```

该分组选择器函数将始终有三个参数，将总是返回一个双精度浮点数值。属于同一组的数据点应该返回同一个值，该值从该函数返回。因为您需要按年进行分组，该函数返回一个新的DateTime类型的值，该值设置为目标年份的第一天。
每一个发生在2014年的数据点将通过该函数返回相同的日期值（作为一个双精度浮点数值），因此被放在同一个分组。



现在，如果你还想通过一个月进行分组的话，那么你只需要修改其中两行代码：

C#

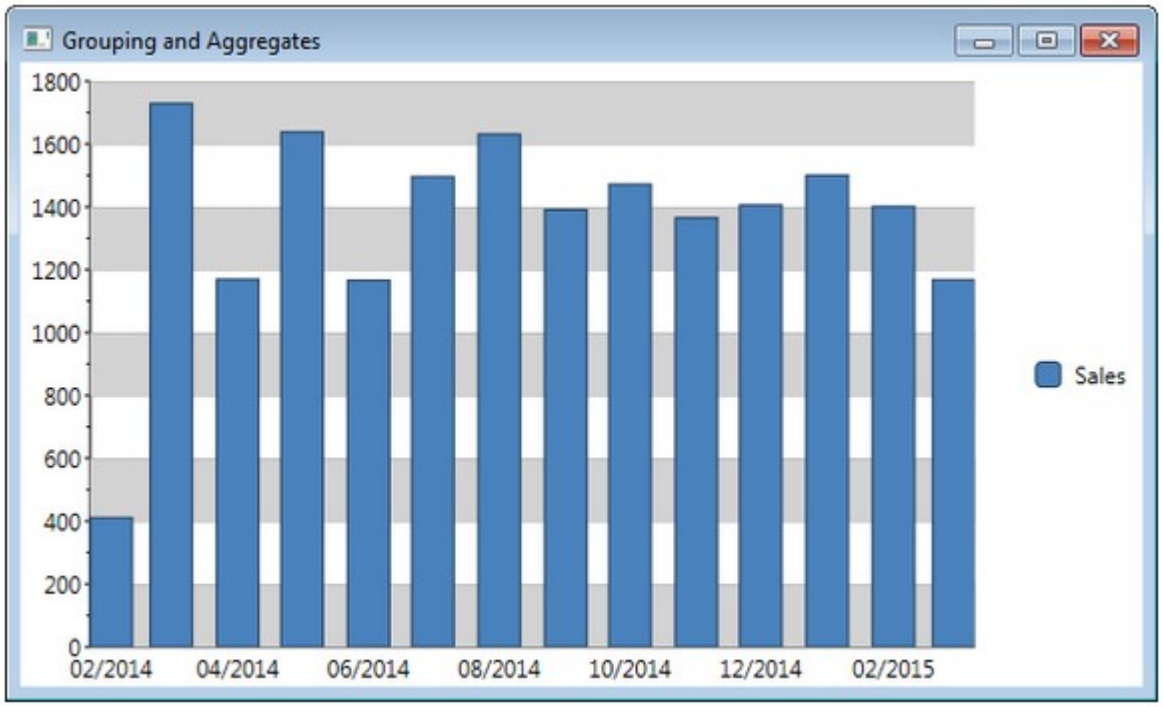
```
double GroupSelectorByDate(double x, double y, object o)

{
// 返回年份为双精度浮点数
DateTime dt = x.FromOADate();
// 为了按照月份进行分组，我们返回该日期的年份和月份值
return new DateTime(dt.Year, dt.Month, 1).ToOADate(); }
```

您也必须要修改AnnoFormat属性以便能够正确显示月份：

C#

```
c1Chart1.View.AxisX.AnnoFormat = "MM/yyyy";
```

由此产生的图表将类似于以下图像：



这里是SampleItem 类，供参考：

```
C#
public class SampleItem
{ public double Value { get; set; } public DateTime Date { get; set; }

}
```

自定义分组

WPF和Silverlight版Chart
允许您通过AggregateGroupSelector属性创建自定义分组以及聚合函数。下面的示例将引导您创建一个自定义的基于分类进行分组的自定义聚合函数。您将会在您的WPF或Silverlight应用程序中的
MainWindow.xaml 页面开始这一过程。
首先，你需要添加一个C1Chart控件至您的应用程序并将其命名为"chart"：

```
XAML
<c1chart:C1Chart Name="chart"></c1chart:C1Chart>
```

然后，添加一个一般的按钮控件，然后设置单击事件如下：

```
XAML
<Button Content="New Data" Width="100" Click="Button_Click" />
```

切换到代码视图。将下面的语句添加到页面顶部：

```
C#
using C1.WPF.C1Chart;
//或者
using C1.Silverlight.Chart;
```

然后，编辑MainWindow()构造函数，它将类似于以下：

```
C#
public MainWindow()
{
InitializeComponent();

CreateSampleChart();
}
```

添加CreateSampleChart()方法：

```
C#
void CreateSampleChart()

{

}
```

在CreateSampleChart()方法中，创建一个列表包含项目名称：

```
C#
var keys = new List<string> { "oranges", "apples", "lemons", "grapes" };
```

然后，添加一个绑定的DataSeries：

```
C#
```

```
for (int i = 0; i < 2; i++)
{ var ds = new DataSeries()
{
ItemsSource = SampleItem.CreateSampleData(40),
ValueBinding = new Binding() { Path = new PropertyPath("Value") },

Aggregate = Aggregate.Sum,
Label = "s" + i
};
```

添加AggregateGroupSelector函数以及向图表添加DataSeries的代码。这里，AggregateGroupSelector函数将从SampleItem类返回项目名称，我们将添加下一步：

C#

```
ds.AggregateGroupSelector = (x, y, o) =>
{
// 来自于类别列表中的索引
return keys.IndexOf(((SampleItem)o).Name);

};
```

chart.Data.Children.Add(ds); } chart.Data.ItemNames = keys; }
Button_Click事件处理将在其调用new之前清除旧数据，每次用户单机此按钮时，将生成新的随机数据：

C#

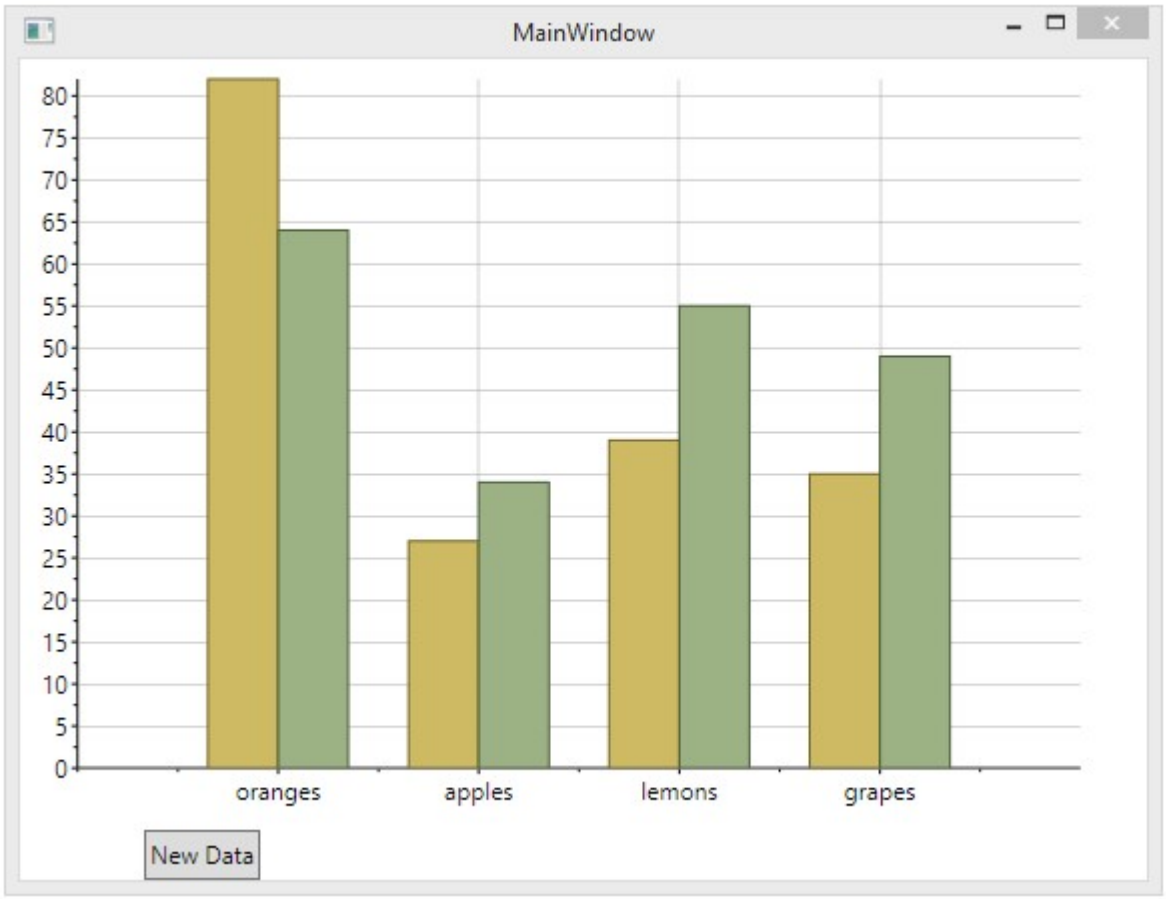
```
private void Button_Click(object sender, RoutedEventArgs e)

{
chart.Data.Children.Clear(); CreateSampleChart();
}
```

最后，添加SampleItem类。这将为您的图表控件创建随机数据：

C#

```
public class SampleItem
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="79247012-0933-4a16-abb1-190ee5e22c5b"><ac:plain-text-body><![CDATA[{ public string Name { get; set; } public
double Value { get; set; } static Random rnd = new Random(); public static SampleItem[] CreateSampleData(int cnt)
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="dbf9aa43-3c81-4526-94c3-100415987554"><ac:plain-text-body><![CDATA[ { var names = new string[] { "oranges", "
apples", "lemons", "grapes" }; var array = new SampleItem[cnt]; for (int i = 0; i < cnt; i++)
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="0480d974-549b-40ba-84d9-04a59fa903b2"><ac:plain-text-body><![CDATA[ { array[i] = new SampleItem() { Value =
rnd.Next(1, 10), Name = names[rnd.Next(names.Length)] }; } return array;
]]></ac:plain-text-body></ac:structured-macro>
}
}
}
}
```

上面的代码和标记在一个应用程序中，将生成类似于下面的图像：



交互

C1Chart包含内置工具，简化了为最终用户交互行为的实现。最终用户可以使用鼠标和换档键的组合来浏览、旋转并缩放图表。交互操作功能的控制中心是C1Chart的Action属性。Action对象具有允许自定义界面的一些属性。所有的属性可以在设计时通过属性窗体的Action集合编辑器或者通过XAML以及编程方式通过Actions集合设置或修改。

下面的代码显示了如何设置Actions属性以其用最终用户交互：

XAML

```
<c1:C1Chart.Actions>
<c1:ZoomAction />
<c1:TranslateAction Modifiers="Shift" /> <c1:ScaleAction Modifiers="Control" />

</c1:C1Chart.Actions>
```

下面的代码展示了如何通过编程方式通过C1Chart.Actions集合设置Actions属性：

C#

```
c1.Chart.Actions.Add(new ZoomAction());
```

下面的列表显示图表支持的操作：

旋转动作允许改变视角。此动作仅适用于带有三维效果的图表。

Rotate3DAction类表示3D图表的旋转行为（仅WPF支持）。

Scale动作沿着选定的坐标轴增加或减小图表显示的范围。ScaleAction 类表示缩放行为。

注意：注意：如果MinScale属性的值为零，则缩放不适用于图表的坐标轴。MinScale 属性指定可以为坐标轴设置的最小缩放值。

偏移行为提供了滚动绘图区的机会。TranslateAction类表示偏移行为。

注意：注意：如果Axis.Scale属性大于 1，您将无法沿着坐标轴平移。

Zoom行为允许用户选择一个方形区域以便查看。

注意：注意：如果MinScale属性的值为零，则缩放不适用于图表的坐标轴。MinScale 属性指定可以为坐标轴设置的最小缩放值。

缩放，平移和缩放仅适用于笛卡尔坐标轴的图表。

运行时的交互式旋转仅在三维图表上可用（仅WPF）。

Actions对象提供了一组属性，以帮助自定义动作的行为。
MouseButton以及Modifiers属性指定可以调用行为的鼠标按钮以及按钮（ALT，CONTROL或者SHIFT）组合。

改变三维图表的旋转角度

该功能仅在该功能仅在WPF下可用。下可用。
为了在运行时改变三维图表类型的旋转视图，向Actions集合添加Rotate3DAction
类。例如，为了使用鼠标中键旋转图表，请使用下面的XAML代码：

XAML

```
<c1chart:C1Chart.Actions>
<c1chart:Rotate3DAction MouseButton="Middle" />

</c1chart:C1Chart.Actions>
```

启用二维图表的运行时交互

缩放，改变范围，以及平移行为由指定的鼠标按钮加上可选的修饰键（Alt|Ctrl|Shift）调用。行为应当被放置在Actions
（在线文档 'Actions 属性'）集合中。下面的XAML标记定义了一组动作。

XAML

```
<c1chart:C1Chart.Actions>
<!-- 使用鼠标左键滚动数据 -->
<c1chart:TranslateAction MouseButton="Left" /> <!-- 通过Ctrl + 鼠标左键改变范围 -->

<c1chart:ScaleAction MouseButton="Left" Modifiers="Ctrl"/>
<!-- 通过Shift + 鼠标左键以缩放选中的方形区域-->
<c1chart:ZoomAction MouseButton="Left" Modifiers="Shift" />
</c1chart:C1Chart.Actions>
```

动作与坐标轴的属性密切相关（Min，Max，Scale，MinScale）。当 Axis.Scale = 1
时，沿着坐标轴的平移操作将不可用。MinScale设置在动作执行过程中，缩放或者改变范围所能达到的限制。
缩放 C1Chart 为了在C1Chart中添加缩放行为，使用一些自定义代码在图表的MouseWheel事件处理中。

C#

```
private void chart_MouseWheel(object sender, MouseWheelEventArgs e)
{ if (Keyboard.Modifiers == ModifierKeys.Control && e.Delta == -120)
{ chart.View.AxisX.Scale += .1; chart.View.AxisY.Scale += .1; } else if (Keyboard.Modifiers == ModifierKeys.Control &&
e.Delta == 120)
{ chart.View.AxisX.Scale -= .1; chart.View.AxisY.Scale -= .1; }
}
```

为了使得用户可以在缩放时在附近移动图表，添加以下内容至C1Chart的XAML标记：

XAML

```
<c1c:C1Chart x:Name="chart" MouseWheel="chart_MouseWheel" >

<c1c:C1Chart.Actions>
<c1c:TranslateAction MouseButton="Left" />
</c1c:C1Chart.Actions>
</c1c:C1Chart>
```

在缩放时修改一个气泡图的范围

为了在缩放时修改一个气泡图的范围，应当在PlotElementLoaded （在线文档 'PlotElementLoaded 事件'）事件中调整范围，如下所示：

C#

```
var ds = new BubbleSeries()
{
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="43c86294-e7ee-47c0-b3cb-2e496ad10d9e"><ac:plain-text-body><![CDATA[XValuesSource = new double[] { 1, 2, 3, 4
},
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="0d9eb853-625b-4ee6-80ae-973c676a5e88"><ac:plain-text-body><![CDATA[ValuesSource = new double[] { 1, 2, 3, 4
},
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="3f45aed8-000a-44a0-937d-d28ca88cd630"><ac:plain-text-body><![CDATA[SizeValuesSource = new double[] { 1, 2, 3,
4 },
]]></ac:plain-text-body></ac:structured-macro>
}; ds.PlotElementLoaded += (s, e) =>
```

```
{ var pe = (PlotElement)s; pe.RenderTransform = new ScaleTransform()
{
ScaleX = 1.0 / chart.View.AxisX.Scale,
ScaleY = 1.0 / chart.View.AxisY.Scale }; pe.RenderTransformOrigin = new Point(0.5, 0.5);
};
chart.Data.Children.Add(ds); chart.ChartType = ChartType.Bubble;
chart.Actions.Add(new TranslateAction()); chart.Actions.Add(new ScaleAction() { Modifiers = ModifierKeys.Control });
```

标记和标签

WPF和Silverlight版Chart对显示绑定可交互的标记和标签提供了特别支持。没有单一的方法来创建或者在图表中显示标记，所以我们的策略是为C1Chart控件提供一个可扩展的对象模型来帮助您创造所需要的精确设置。
本主题将涵盖ChartPanelObject 以及ChartView.Layers集合以及您如何使用这些在图表总提供一个定制的形形色色的标记和标签。
为了在图表中使用图表面板，首先必须向ChartView的Layers集合中添加面板：

XAML

```
<c1chart:C1Chart x:Name="chart">

<c1chart:C1Chart.View>
<c1chart:ChartView>
<c1chart:ChartView.Layers>
<c1chart:ChartPanel >
<!-- ChartPanelObjects -->

</c1chart:ChartPanel>
</c1chart:ChartView.Layers>
</c1chart:ChartView>
</c1chart:C1Chart.View>
</c1chart:C1Chart>
```

通过ChartView.Layers集合，您可以添加任意数量的ChartPanel。每一个面板可以具有任意数量的ChartPanelObject，每一个对象是一个基础的UI元素，用于定义我们的标记。ChartPanelObject有几个关键属性：
Attach – 设置是否该对象为关联或者"吸附"到数据点。你可以附加到X，Y，同时或者不关联。
Action – 设置交换行为，比如说鼠标移动，鼠标拖拽或者没有交互行为。
DataPoint
– 显式地设置初始的数据点，或者创建一个静态的标记。您可以设置ChartPanelObject.Content属性为任何UIElement。这允许您定义您的标记的外观并提供绑定到数据点。您还
可以使用Alignment属性以帮助定义您的标记的外观，–
您可以创建一个居中显示的标记。在这种情况下，您最好请设置HorizontalAlignment属性为"Center"。
下面的XAML定义左下角位于数据坐标 X= 0，Y = 0的文本标签：

XAML

```
<c1chart:ChartPanelObject DataPoint="0,0" VerticalAlignment="Bottom">
<TextBlock Text="Zero"/>
</c1chart:ChartPanelObject>
```

注意：注意：没有必要同时指定两个坐标。如果坐标设置为double.Nan那么元素没有特定的X或Y坐标。

我们可以创建水平标记在y= 0的位置。请注意，HorizontalAlignment 属性设置为Stretch，元素填充绘图区的宽度。

XAML

```
<!-- 水平线 -->
<c1chart:ChartPanelObject DataPoint="NaN,0"
HorizontalAlignment="Stretch">
<Border BorderBrush="Red" BorderThickness="0,2,0,0"

Margin="0,-1,0,0" />
</c1chart:ChartPanelObject>
```

下面的示例创建一个垂直标记：

XAML

```
<!-- 垂直线 -->
<c1chart:ChartPanelObject DataPoint="0,NaN" VerticalAlignment="Stretch">

<Border BorderBrush="Red" BorderThickness="2,0,0,0"
Margin="-1,0,0,0" />
</c1chart:ChartPanelObject>
```

注意：注意：图表面板对象仅支持主坐标轴。对于辅助轴，您需要执行坐标转换。

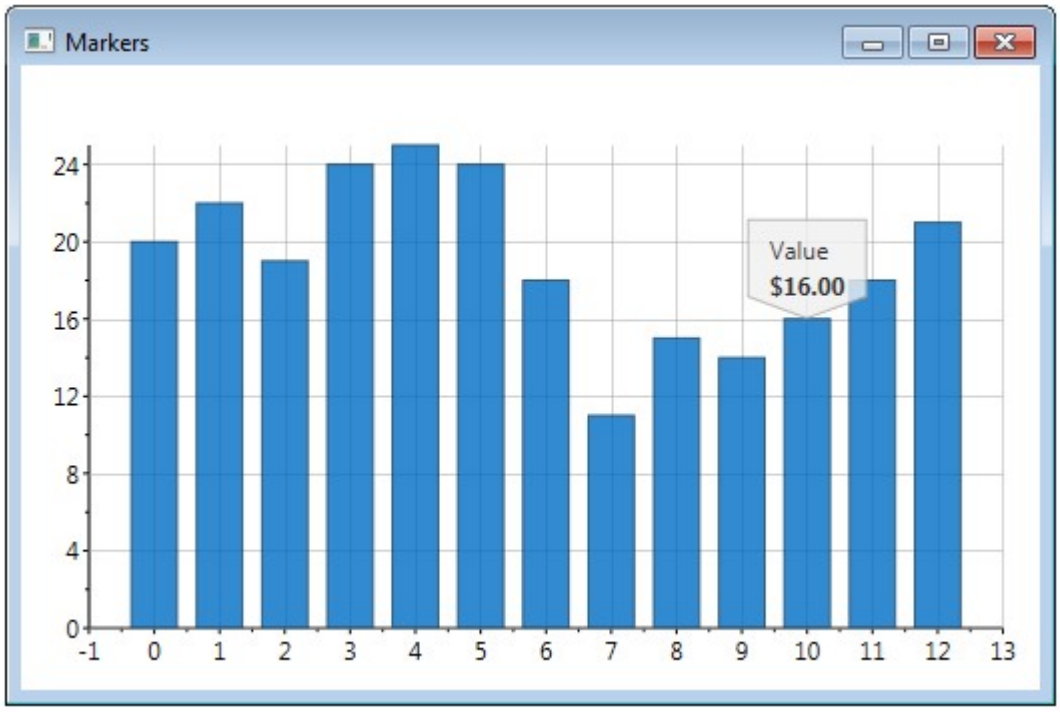
简单绑定标记你可以通过设置五个属性来创建一个简单的绑定标记。下面的XAML标记展示如何完成这个操作：

XAML

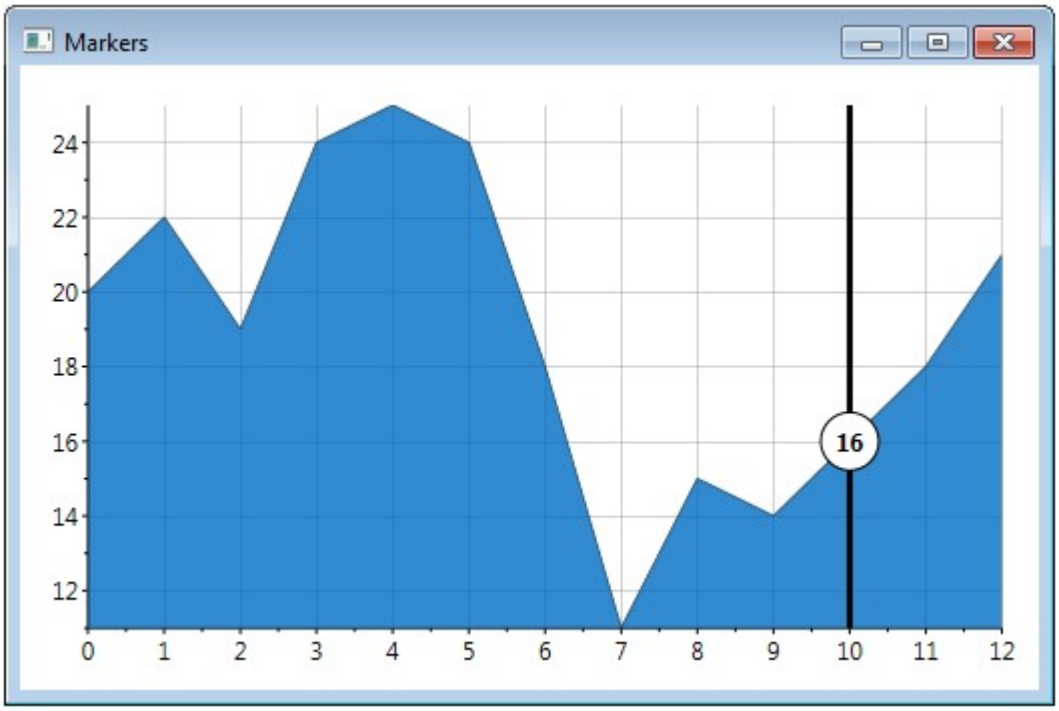
```
<!-- 简单绑定标记 -->
<c1:ChartPanelObject x:Name="obj" Attach="DataX"

Action="MouseMove"
DataPoint="-1,-1"
HorizontalAlignment="Center"
VerticalAlignment="Top" Width="60" Height="50">
<c1:ChartPanelObject.RenderTransform>
<TranslateTransform Y="-50"/>
</c1:ChartPanelObject.RenderTransform>
```

```
<Grid DataContext="{Binding RelativeSource={x:Static
RelativeSource.Self},Path=Parent}" Opacity="0.8">
<Path Data="M0.5,0.5 L23,0.5 23,23 11.61165,29.286408 0.5,23 z"
Stretch="Fill" Fill="#FFF1F1F1" Stroke="DarkGray" StrokeThickness="1"/>
<StackPanel VerticalAlignment="Center" HorizontalAlignment="Center">
<TextBlock Text="Value" Margin="2 0"/>
<TextBlock x:Name="label" Text="{Binding DataPoint.Y, StringFormat=c2}"
FontWeight="Bold" Margin="2"/>
</StackPanel>
</Grid> </c1:ChartPanelObject>
```

请注意，您设置了以下属性：
Attach = DataX
Action = MouseMove
DataPoint = -1,-1
HorizontalAlignment = Center VerticalAlignment = Top
该应用程序运行起来将类似于以下图像：



线和点标记
对于一个面积图或折线图，您可能希望有一个标记，标记X-或Y-轴以及数据点。对于可视化参考，一个线或者点标记将类似于以下的图像所示：



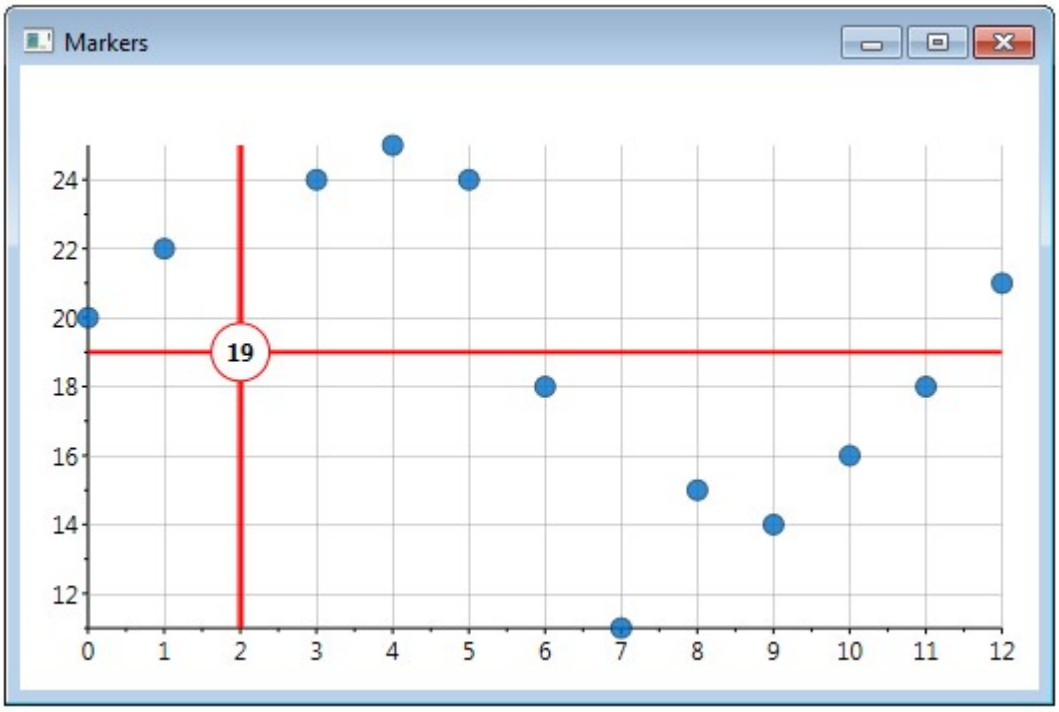
要创建这样的一个标记，需要设置的最重要的属性之一是VerticalAlignment
属性。当您设置该属性的值为"Stretch"时，标记将拉伸至整个绘图区高度，以提供垂直线。下面是您将在标记中设置的属性：
Attach = DataX
Action = MouseMove
DataPoint = -1, NaN
HorizontalAlignment = Center
VerticalContentAlignment = Stretch
请注意，您正在设置数据点的Y部分的值为NaN。这也将有助于设置完全的的垂直线，因为标记永远不会附加到一个特定的数据点。上图中的圆形标签是另一个ChartPanelObject 对象，该对象将放置在绘图区元素上。它的DataPoint属性将被设置为非NaN的值。
您可以只使用XAML标记创建这种效果，完全不需要添加任何代码！

XAML

```
<!-- 垂直线以及点标记 -->
<c1:ChartPanelObject x:Name="vline"
Attach="DataX"
Action="MouseMove"
DataPoint="-1, NaN"
VerticalContentAlignment="Stretch"
HorizontalAlignment="Center">
<Border Background="Black" BorderBrush="Black" Padding="1" BorderThickness="1
0 0 0" />
</c1:ChartPanelObject>
<c1:ChartPanelObject x:Name="dot"
Attach="DataX"
Action="MouseMove"
DataPoint="0.5,0.5"
HorizontalAlignment="Center"
VerticalAlignment="Center">
<Grid DataContext="{Binding RelativeSource={x:Static
RelativeSource.Self},Path=Parent}">
<Ellipse Fill="White" Stroke="Black" StrokeThickness="1" Width="30"
Height="30" />
<TextBlock x:Name="label" Text="{Binding DataPoint.Y, StringFormat=n0}"
FontWeight="Bold" VerticalAlignment="Center" HorizontalAlignment="Center"/> </Grid>
</c1:ChartPanelObject>
```

十字线标记

对于一些图表，你可能想要一个自由浮动的带有十字线的标记，被设计用来强调某一个数据点。在本主题中，您会在线形和点标记的基础上添加一个水平的标记。完成后的带有标记的图表将类似以下图像：



下面的XAML，您会再一次设置数据点为NaN：

XAML

```
<!-- crosshairs -->
<c1:ChartPanelObject x:Name="vline"
Attach="None"
Action="MouseMove"
DataPoint="-1, NaN"
VerticalContentAlignment="Stretch"
HorizontalAlignment="Center">
<Border Background="Red" BorderBrush="Red" Padding="1" BorderThickness="1 0 0
0" />
</c1:ChartPanelObject>
<c1:ChartPanelObject x:Name="hline"
Attach="None"
Action="MouseMove"
DataPoint="NaN, -1"
HorizontalContentAlignment="Stretch"
VerticalAlignment="Center">
<Border Background="Red" BorderBrush="Red" Padding="1" BorderThickness="0 1 0 0"

/>
</c1:ChartPanelObject>
<c1:ChartPanelObject x:Name="dot"
Attach="None"
Action="MouseMove"
DataPoint="0.5,0.5"
HorizontalAlignment="Center"
VerticalAlignment="Center">
<Grid DataContext="{Binding RelativeSource={x:Static
RelativeSource.Self},Path=Parent}">
<Ellipse Fill="White" Stroke="Red" StrokeThickness="1" Width="30" Height="30" />

<TextBlock x:Name="label" Text="{Binding DataPoint.Y, StringFormat=n0}"
FontWeight="Bold" VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</c1:ChartPanelObject>
```

在代码中添加标记

以上所有的主题描述如何使用XAML标记添加一个标记。您可能有一个项目需要在代码中添加一个标记。
首先，你需要创建一个新的ChartPanel：

C#

```
var pnl = new ChartPanel();
```

一旦您添加了一个新的ChartPanel，您将添加一个新的ChartPanelObject并设置其对齐属性：

```
C#
var obj = new ChartPanelObject()
```

```
{
HorizontalAlignment = HorizontalAlignment.Right,
VerticalAlignment = VerticalAlignment.Bottom };
```

下一步，您将添加一个边框元素：

```
C#
var bdr = new Border()
{
Background = new SolidColorBrush(Colors.Green) { Opacity = 0.4 },
BorderBrush = new SolidColorBrush(Colors.Green),
BorderThickness = new Thickness(1, 1, 3, 3), CornerRadius = new CornerRadius(6, 6, 0, 6),

Padding = new Thickness(3)
};
```

添加一个包含两个TextBlock控件的StackPanel元素。请注意，绑定源是您的ChartPanelObject：

```
C#
var sp = new StackPanel();
var tb1 = new TextBlock(); var bind1 = new Binding(); bind1.Source = obj; bind1.StringFormat = "x={0:#.##}"; bind1.Path =
new PropertyPath("DataPoint.X"); tb1.SetBinding(TextBlock.TextProperty, bind1);
var tb2 = new TextBlock(); var bind2 = new Binding(); bind2.Source = obj; bind2.StringFormat = "y={0:#.##}"; bind2.Path =
new PropertyPath("DataPoint.Y"); tb2.SetBinding(TextBlock.TextProperty, bind2);
sp.Children.Add(tb1); sp.Children.Add(tb2); bdr.Child = sp;
```

设置ChartPanelObject的Content，DataPoint，以及Aciton属性，接下来添加该ChartPanelObject至ChartPanel。最后一行代码将图层集合添加到图表控件中。

```
C#
obj.Content = bdr; obj.DataPoint = new Point(); obj.Action = ChartPanelAction.MouseMove; pnl.Children.Add(obj);
```

```
chart.View.Layers.Add(pnl);
```

最后一行代码，您需要设置Attach属性：

```
C#
obj.Attach = ChartPanelAttach.MouseMove; };

}
}
```

在这个主题中，您已经创建了一个图表标记，将跟随您的鼠标移动。

通过代码更新标签

您还可以使用代码来更新窗体上的标签元素。例如，您可能希望创建一个标记，它将在图表外更新一个标签。为了达到这个目的，你需要监听标记上的DataPointChanged 事件。

以下代码获取标记的数据点的值并设置该值给您窗体上的某个TextBlock：

```
C#
private void ChartPanelObject_DataPointChanged(object sender, EventArgs e)

{
//通过代码从标记更新标签
var obj = (ChartPanelObject)sender; if (obj != null)
{ lbl.Text = obj.DataPoint.Y.ToString("c2");
}
}
```