WPF及Silverlight版本Chart

WPF及Silverlight版Chart通过强大的渲染系统，丰富的带有样式的元素，动画以及数据绑定能力给图表的表示带来革命性的变化。

在WPF中，您可以在一个智能客户端应用程序或者利用XBAP支持的强项将程序部署到Web平台。

通过使用内置的主题，图表类型以及多种多样的调色板，您定制化一个图表的过程只需要轻点鼠标即可完成，完全不需要额外的编码工作。仅需一步设置，就可以转换您的数据并添加专业级的图表至您的应用程序；WPF及Silverlight版Chart支持全部的热门图表类型，包括条形图，柱状图，折线图，面积图，饼图以及更多其他图表类型。今天，把你的数据可视化更上一层楼！

入门

接下来带您开始了解和学习Chart控件。

主要特性

WPF及Silverlight版Chart包含以下独有功能：

- **超过40种的图表类型**
  可以为您的Silverlight应用程序从30种常用的2D图表类型选择您所需要的图表展示。可用的图表类型包括折线图，散点图，饼图，面积图以及更多其他种类的图表。

- **仅需设置两个属性就可以得到专业的设计**
  图表包含12种内置的主题以及22种内置的调色板。仅需在Visual Studio中设置一个属性即可轻松完成设置！主题将应用到整个图表区域，而调色板仅应用于图表元素（条形，点，饼图块，等等）。将主题和不同的调色板组合，可以毫不费力地创建无尽的外观组合。

- **标签以及工具提示**
  可以通过标签或者工具提示形式在图表元素上显示相关联的数据的值。任意的UI元素可以用作标签和工具提示，并允许完全定制。

- **图表图例**
  仅需设置一个属性，即可通过C1ChartLegend控件创建一个关联到图表的独立的图表图例。该设计在排布图例区以及设置图例区样式时，提供了最大限度的灵活性。

- **完全可交互式图表**
  通过允许放大/缩小，拉伸以及滚动图表，极大的提高了最终用户所获得的用户体验。

- **多重坐标轴**
  图表支持多重，相关联的坐标轴，只需要简单地定义一个Axis对象并添加到图表的View.Axes集合，即可将这些坐标轴添加到图表。

- **对数轴刻度**
  图表支持任意底数的对数坐标轴刻度。

- **趋势线**
  可以通过趋势线分析图表中的数据走势。图表支持几种不同的自动趋势线，包括多项式，指数，对数，乘幂，傅里叶，平均值，最小值以及最大值。

- **高亮和阴影效果**
  创建具有高亮效果的边框，并可以在绘图区元素之后添加不同柔和程度的阴影。

- **堆叠图表**

堆叠图表为展示复杂数据提供了一种简单的方式。折线图，面积图，条形图，雷达图以及地块图可以堆叠在一起以便在有限的空间内显示更加复杂的数据。

- **动态图形**
  图表采用了Silverlight平台下可用的动态图形的优势，包括透视和动画。
- **支持Silverlight Toolkit 主题（仅Silverlight）**
  除了12种内置的主题之外，图表还附带了最流行的Microsoft Silverlight Toolkit主题，包括ExpressionDark，ExpressionLight，WhistlerBlue，RainerOrange，ShinyBlue，以及BureauBlack。
- **XBAP支持（仅WPF）**
  在WPF版本中，C1Chart完全兼容XBAP发布功能。XBAP发布方式允许将全部功能的应用程序发布到支持的客户端浏览器，无须使用Windows安装程序。

快速入门

请参见以下步骤：

第一步：添加一个图表至您的工程

在此步骤中，您将开始使用WPF及Silverlight版Chart在Visual Studio 中创建一个图表应用。当您添加C1Chart（在线文档'C1Chart 类'）控件至您的Visual Studio工程，您将看到一个具有功能的带有假数据的柱状图。完成以下步骤：

1. 在 Visual Studio 中创建一个新的 WPF 或者 Silverlight 应用程序。
    1. 选择 File | New | Project。将显示新建工程对话框。
    2. 在此新建工程对话框中，在位于左侧的面板中选择一种语言，并选择一个模版。
        1. 对于WPF应用程序，选择 Windows Desktop。接下来在中间的面板中选择WPF应用程序。
        2. 对于Silverlight程序，选择Silverlight。接下来从中间的面板中选择 Silverlight 应用程序。
        3. 为您的应用程序命名，并选择OK。您的应用程序将创建并打开。
2. 通过右键单击 Reference 文件夹并选择 Add Reference，向您的应用程序添加引用。将打开 Reference Manager。
    1. 浏览并定位到适合您工程的程序集引用。
        1. 对于WPF应用程序，添加以下引用：C1.WPF.Chart.4.dll
        2. 对于 Silverlight 应用程序，添加以下引用：C1.Silverlight.Chart.5.dll
3. 在您的 <Window> 或者 <UserControl> 标签中添加以下命名空间声明。如果您在操作XAML标记语言，这将在接下来允许您在该工程中添加图表数据。

```
XAML
xmlns:System="clr-namespace:System;assembly=mscorlib"
```

4. 将光标移动到位于Window或者UserControl内部的<Grid></Grid>标签之间。具体是Window还是UserControl取决于您所创建的应用程序类型。
5. 在Visual Studio 的工具栏中找到 C1Chart 控件。双击该控件以将其添加至您的应用程序。XAML标记语言将重新生成如下所示的代码：

```
XAML
<Grid x:Name="LayoutRoot">
<c1chart:C1Chart></c1chart:C1Chart>
</Grid>
```

6. 为您的图表命名，之后您可以通过代码对其进行访问。您的标记语言将类似于以下的代码示例：

XAML

```
<c1chart:C1Chart Margin="0,0,8,8" MinHeight="160" MinWidth="240"
Name="c1Chart1">
</c1chart:C1Chart>
```

您所完成的部分

到此您已经成功地创建了包含一个 C1Chart 控件的 WPF 或者 Silverlight 应用程序。在下一步（第二步：向图表添加数据）中，您将为 C1Chart 添加数据。

第二步：向图表添加数据

在上一步中，您完成了向窗体添加添加了C1Chart（在线文档'C1Chart 类'）控件的工作。在这一步，您将为其添加一个DataSeries（在线文档'DataSeries 类'）对象以及数据。有两种方式添加一个DataSeries：通过XAML标记语言或者通过代码进行添加。下面的选项卡包含了全部的两种向图表添加数据的方式。选择合适的选项卡并完成其中的步骤：

XAML

1. 通过编辑<c1chart:C1Chart>标签设置ChartType，使得结果类似下面的代码：

XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="Bar" Margin="0,0,8,8"
MinHeight="160" MinWidth="240" Content="C1Chart">
</c1chart:C1Chart>
```

2. 在XAML中，您可以通过使用C1ChartData对象添加您的数据：

XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="Bar" Margin="0,0,8,8"
MinHeight="160" MinWidth="240" Content="C1Chart">
<c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:ChartData.ItemNames>
                        <x:Array Type="{x:Type System:String}">
                            <System:String>Hand Mixer</System:String>
                            <System:String>Stand Mixer</System:String>
                            <System:String>Can Opener</System:String>
                            <System:String>Toaster</System:String>
                            <System:String>Blender</System:String>
                            <System:String>Food Processor</System:String>
                            <System:String>Slow Cooker</System:String>
                            <System:String>Microwave</System:String>
                        </x:Array>
                    </c1chart:ChartData.ItemNames>
                    <c1chart:DataSeries Values="80 400 20 60 150 300 130 500"
AxisX="Price" AxisY="Kitchen Electronics" Label="Price"/>
```

```
                </c1chart:ChartData>
</c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

在本步骤中，我们使用了一个具有八个X值的DataSeries。我们向ChartData添加了字符串类型的ItemName以表示每一个数据值的字符串名称。我们使用了一个字符串名称的数组表示ItemNames，因为其中有个别项目的名称包含空格。我们可以使用System:String命名空间，因为我们曾经在第一步中声明了该命名空间。向您的工程添加图表。

代码

1. 右键单击MainPage.xaml文件，并选择代码视图。
2. 直接添加C1.Silverlight.C1Chart命名空间

Visual Basic

```
Imports C1.Silverlight.Chart
```

C#

```
using C1.Silverlight.Chart;
```

3. 在Windows1类的构造器中添加以下代码以创建一个条形图：

Visual Basic

```
' 清除之前的旧数据
c1Chart1.Data.Children.Clear()
' 添加数据
Dim ProductNames As String() = {"Hand Mixer", "Stand Mixer", "Can Opener",
"Toaster", "Blender", "Food Processor", "Slow Cooker", "Microwave"}
Dim PriceX As Integer() = {80, 400, 20, 60, 150, 300, 130, 500}
' 为产品价格创建单一系列
Dim ds1 As New DataSeries()
ds1.Label = "Price X"
'set price data
ds1.ValuesSource = PriceX
' 添加系列至图表
c1Chart1.Data.Children.Add(ds1)
' 添加项目名称
c1Chart1.Data.ItemNames = ProductNames
' 设置图表类型
c1Chart1.ChartType = ChartType.Bar
```

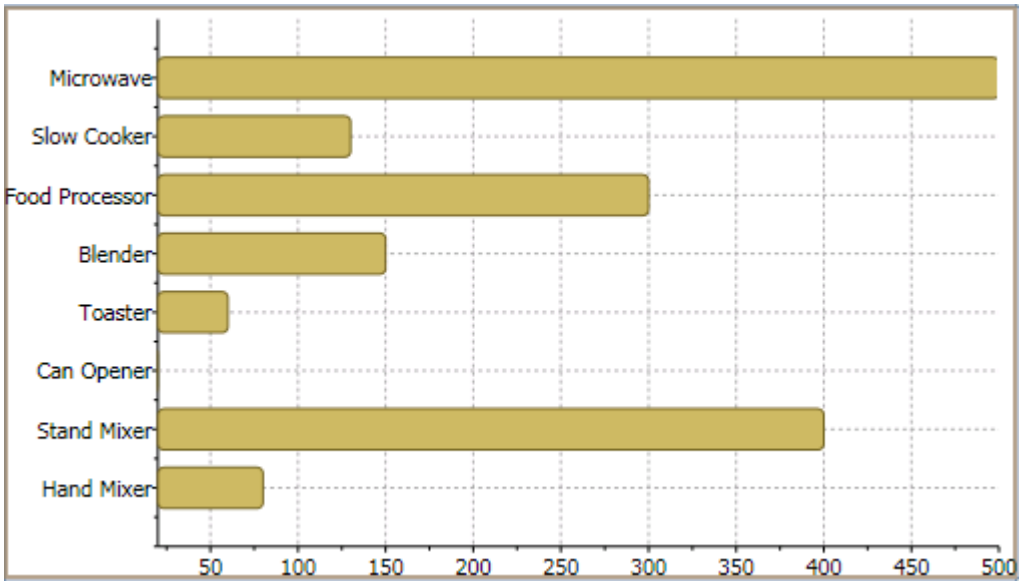
C#

```
// 清除之前的旧数据
```

```
    c1Chart1.Data.Children.Clear();
// 添加数据
    string[] ProductNames = { "Hand Mixer", "Stand Mixer", "Can Opener",
"Toaster", "Blender", "Food Processor", "Slow Cooker", "Microwave" };
    int[] PriceX = { 80, 400, 20, 60, 150, 300, 130, 500 };

// 为产品价格创建单一系列
    DataSeries ds1 = new DataSeries();
    ds1.Label = "Price X";
//设置价格数据
    ds1.ValuesSource = PriceX;

// 添加系列至图表
    c1Chart1.Data.Children.Add(ds1);

// 添加项目名称
    c1Chart1.Data.ItemNames = ProductNames;
// 设置图表类型
    c1Chart1.ChartType = ChartType.Bar;
```

您所完成的部分

至此您已经成功地向C1Chart添加数据，因此当运行您的应用程序时，Y轴将出现字符串值，如下图所示：



第三步：格式化坐标轴

在本步骤中，您将添加一个ChartView（在线文档'ChartView 类'）对象，您可以通过该对象自定义X轴。您可以通过XAML标记语言或者代码完成此步骤。从下方选择适当的选项卡并完成指定步骤：

In XAML

1. 添加ChartView对象，之后您可以设置X-轴以及Y-轴的标题
ChartView对象表示图表中包含数据（包括坐标轴）的区域。关于图表坐标轴的更多信息，请参见坐标轴。坐标轴标题是UIElement对象，而不仅仅是简单的文本。在该示例中，我们将使用TextBlock元素为X-轴和Y-轴的标题指定文本内容。在我们添加了TextBlock元素之后，我们接下来可以做一些格式化工作，比如数修改其前景色并将文本对其设置为居中。

XAML

```
<c1chart:C1Chart >
      <c1chart:C1Chart.View>
        <c1chart:ChartView>
          <c1chart:ChartView.AxisX>
            <c1chart:Axis>
              <c1chart:Axis.Title>
                <TextBlock Text="Price" TextAlignment="Center"
Foreground="Crimson"/>
              </c1chart:Axis.Title>
            </c1chart:Axis>
          </c1chart:ChartView.AxisX>
          <c1chart:ChartView.AxisY>
            <c1chart:Axis>
              <c1chart:Axis.Title>
                <TextBlock Text="Kitchen Electronics" TextAlignment="Center"
Foreground="Crimson"/>
              </c1chart:Axis.Title>
            </c1chart:Axis>
          </c1chart:ChartView.AxisY>
        </c1chart:ChartView>
      </c1chart:C1Chart.View>
    </c1chart:C1Chart>
```

2. 设置X-轴的值从零开始，并将默认的Axis.MajorUnit单位值从50修改为20。同时设置AutoMin属性的值为False，因此我们可以让值从零开始计算的设置生效，否则坐标轴仍然会使用最小的数据值。至此，针对View对象的XAML代码应当如下面所示：

XAML

```
<c1chart:C1Chart.View>
                <c1chart:ChartView>
                    <c1chart:ChartView.AxisX>
                        <c1chart:Axis Min="0" Max="500" MajorUnit="20"
AutoMin="False">
                            <c1chart:Axis.Title>
                                <TextBlock Text="Price"
TextAlignment="Center" Foreground="Crimson" />
                            </c1chart:Axis.Title>
                        </c1chart:Axis>
                    </c1chart:ChartView.AxisX>
                    <c1chart:ChartView.AxisY>
                        <c1chart:Axis>
```

```
                            <c1chart:Axis.Title>
                                <TextBlock Text="Kitchen Electronics"
TextAlignment="Center" Foreground="Crimson" />
                            </c1chart:Axis.Title>
                        </c1chart:Axis>
                    </c1chart:ChartView.AxisY>
                </c1chart:ChartView>
            </c1chart:C1Chart.View>
```

3. 在ChartView.AxisX对象的<c1chart:Axis></c1chart:Axis>标记中间，设置AnnoFormat以改变沿着X-轴显示的值从数值格式为货币格式，设置AnnoAngle属性以旋转X-轴标注为逆时针旋转60度。

XAML

```
<c1chart:Axis AnnoFormat="c" AnnoAngle="60"  />
```

4. 在ChartView.AxisY对象的<c1chart:Axis></c1chart:Axis> 标记中间设置Reversed属性以反向显示Y-轴的值。

In Code

在Public MainPage（）类中添加以下代码以格式化图表坐标轴：

Visual Basic

```
' 设置坐标轴标题
   C1Chart1.View.AxisY.Title = New TextBlock(New Run("Kitchen Electronics"))
   C1Chart1.View.AxisX.Title = New TextBlock(New Run("Price"))

   ' 设置坐标轴范围
   C1Chart1.View.AxisX.Min = 0
   C1Chart1.View.AxisX.Max = 500
   c1Chart1.View.AxisX.MajorUnit = 20

   ' 金融货币格式
   C1Chart1.View.AxisX.AnnoFormat = "c"

   ' 坐标轴标注旋转
   C1Chart1.View.AxisX.AnnoAngle = "60"
```

C#

```
// 设置坐标轴标题
   c1Chart1.View.AxisY.Title= new TextBlock() { Text = "Kitchen Electronics" };
   c1Chart1.View.AxisX.Title = new TextBlock() { Text = "Price" };

   // 设置坐标轴范围
   c1Chart1.View.AxisX.Min = 0;
   c1Chart1.View.AxisX.Max = 500;
   c1Chart1.View.AxisX.MajorUnit = 20;
```

```
// 金融货币格式
c1Chart1.View.AxisX.AnnoFormat = "c";

// 坐标轴标注旋转
c1Chart1.View.AxisX.AnnoAngle=60;
```

下一步，第四步：调整图表的外观，您将学习如何通过编程方式自定义图表的外观。

第四步：调整图表的外观

在此最后一步中，您将通过Theme属性调整图表的外观。

为了通过XAML在Visual Studio中设置图表的主题：

为了明确地在您的图表中定义 Office2007Blue主题，添加以下的Theme标记至<c1chart:C1Chart>标签，之后代码将如下所示：

XAML

```
<c1chart:C1Chart Margin="0,0,8,8" MinHeight="160" MinWidth="240" Content="C1Chart"
ChartType="Bar" Theme="Office2007Blue">
```

为了通过代码在 Visual Studio 中设置图表的主题：

WPF

Visual Basic

```
C1Chart1.Theme = TryCast(C1Chart1.TryFindResource(New
ComponentResourceKey(GetType(C1Chart), "Office2007Blue")), ResourceDictionary)
```

C#

```
C1Chart1.Theme = c1Chart1.TryFindResource( new
ComponentResourceKey(typeof(C1.WPF.C1Chart.C1Chart),   "Office2007Blue")) as
ResourceDictionary;
```

Silverlight

Visual Basic

```
c1Chart1.Theme = ChartTheme.
```

C#

```
c1Chart1.Theme = ChartTheme.Office2007Blue;
```

至此，您完成了：

Office 2007 Blue 主题应用到了 C1Chart 控件。
恭喜完成！您已经完成了WPF版Chart快速入门，在这里您创建了一个图表应用程序，向图表添加了数据，设置了坐标轴的范围，格式化显示了坐标轴的标注，并自定义了图表的外观。

对于WPF版Chart的重点提示

针对WPF以及Silverlight版Chart，以下几个常用提示将在您使用C1Chart控件时对您有所帮助。

提示技巧一：使用BeginUpdate()/EndUpdate()方法以提高性能

这将极大的提高性能，因为重绘动作仅仅在调用了EndUpdate()方法之后进行一次。
例如：

Visual Basic

```
' 开始批量更新
  C1Chart1.BeginUpdate()

  Dim nser As Integer = 10, npts As Integer = 100
  For iser As Integer = 1 To nser
    ' 创建数据数组
    Dim x(npts - 1) As Double, y(npts - 1) As Double
    For ipt As Integer = 0 To npts - 1
      x(ipt) = ipt
      y(ipt) = (1 + 0.05 * iser) * Math.Sin(0.1 * ipt + 0.1 * iser)
    Next
    ' 创建数据系列
    Dim ds = New XYDataSeries()
    ds.XValuesSource = x
    ds.ValuesSource = y
    C1Chart1.Data.Children.Add(ds)
  Next

  ' 设置图表类型
  C1Chart1.ChartType = ChartType.Line

  ' 结束批量更新
  C1Chart1.EndUpdate()
```

C#

```
// 开始批量更新
  c1Chart1.BeginUpdate();
```

```
int nser = 10, npts = 100;
for (int iser = 0; iser < nser; iser++)
{
  // 创建数据数组
  double[] x = new double[npts], y = new double[npts];
  for (int ipt = 0; ipt < npts; ipt++)
  {
    x[ipt] = ipt;
    y[ipt] = (1 + 0.05 * iser) * Math.Sin(0.1 * ipt + 0.1 * iser);
  }

  // 创建数据系列
  XYDataSeries ds = new XYDataSeries();
  ds.XValuesSource = x; ds.ValuesSource = y;
  c1Chart1.Data.Children.Add(ds);
}

// 设置图表类型
c1Chart1.ChartType = ChartType.Line;

// 结束批量更新
c1Chart1.EndUpdate();
```

提示技巧二：对于大量的数据数组，使用折线图或者面积图

当您需要展示大量数据值时，折线图和面积图提供了最优的展示性能。

为了获取更佳的性能，通过设置attached属性，LineAreaOptions.OptimizationRadius启用大量数据展示优化。例如：

Visual Basic

```
LineAreaOptions.SetOptimizationRadius(C1Chart1, 1.0)
```

C#

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, 1.0);
```

强烈建议您使用比较小的值作为范围，比如说1.0~2.0。一个较大的值将影响绘图的精度。

提示技巧三：使用DataSeries.PlotElementLoaded事件更新绘图区元素的外观和行为

当任意绘图区元素（条形，柱形或者饼图，等等）加载完成时，将出发PlotElementLoaded（在线文档'PlotElementLoaded事件'）事件。在此事件中，您可以访问此绘图区元素，以及其关联的数据点。
以下代码按照y-轴的值的不同设置数据点的颜色。例如：

Visual Basic

```
' 创建数据数组
  Dim npts As Integer = 100
  Dim x(npts - 1) As Double, y(npts - 1) As Double
  For ipt As Integer = 0 To npts - 1
    x(ipt) = ipt
    y(ipt) = Math.Sin(0.1 * ipt)
  Next
```

```
  ' 创建数据系列
  Dim ds = New XYDataSeries()
  ds.XValuesSource = x
  ds.ValuesSource = y

  '设置事件处理器
  AddHandler ds.PlotElementLoaded, AddressOf PlotElement_Loaded

  ' 添加数据系列至图表
  C1Chart1.Data.Children.Add(ds)

  ' 设置图表类型
  C1Chart1.ChartType = ChartType.LineSymbols

...

  ' event handler
  Sub PlotElement_Loaded(ByVal sender As Object, ByVal args As EventArgs)
    Dim pe = CType(sender, PlotElement)
    If Not TypeOf pe Is Lines Then
      Dim dp As DataPoint = pe.DataPoint
      ' 正规化 y-值（范围从 0 到 1）
      Dim nval As Double = 0.5 * (dp.Value + 1)
      ' 填充色从蓝色（－1）变化到红色（＋1）
      pe.Fill = New SolidColorBrush(Color.FromRgb(CByte(255 * nval), _
       0, CByte(255 * (1 - nval))))
    End If
  End Sub
```

C#

```
// 创建数据数组
  int npts = 100;
  double[] x = new double[npts], y = new double[npts];
  for (int ipt = 0; ipt < npts; ipt++)
  {
    x[ipt] = ipt;
    y[ipt] = Math.Sin(0.1 * ipt);
  }

  // 创建数据系列
  XYDataSeries ds = new XYDataSeries();
  ds.XValuesSource = x; ds.ValuesSource = y;

  //设置事件处理器
  ds.PlotElementLoaded += (s, e) =>
    {
      PlotElement pe = (PlotElement)s;
      if (!(pe is Lines)) // skip lines
      {
        DataPoint dp = pe.DataPoint;
```

```
        // 正规化 y-值（范围从 0 到 1）
        double nval = 0.5*(dp.Value + 1);

        // 填充色从蓝色（－1）变化到红色（＋1）
        pe.Fill = new SolidColorBrush(
          Color.FromRgb((byte)(255 * nval), 0, (byte)(255 * (1-nval))));
      }
    };

  // 添加数据系列至图表
  c1Chart1.Data.Children.Add(ds);

  // 设置图表类型
  c1Chart1.ChartType = ChartType.LineSymbols;
```

提示技巧四：数据点标签及工具提示

为创建一个数据点标签或工具提示，您应当设置模版至PointLabelTemplate或者PointToolTipTemplate属性。以下示例代码将演示如何显示每一个数据点的索引。

XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="XYPlot">
  <c1chart:C1Chart.Data>
    <c1chart:ChartData>
      <!-- source collection -->
      <c1chart:ChartData.ItemsSource>
        <PointCollection>
          <Point X="1" Y="1" />
          <Point X="2" Y="2" />
          <Point X="3" Y="3" />
          <Point X="4" Y="2" />
          <Point X="5" Y="1" />
        </PointCollection>
      </c1chart:ChartData.ItemsSource>

      <c1chart:XYDataSeries SymbolSize="16,16"
        XValueBinding="{Binding X}" ValueBinding="{Binding Y}">
        <c1chart:XYDataSeries.PointLabelTemplate>
          <DataTemplate>
            <!-- display point index at the center of point symbol  -->
            <TextBlock c1chart:PlotElement.LabelAlignment="MiddleCenter"
                       Text="{Binding PointIndex}" />
          </DataTemplate>
        </c1chart:XYDataSeries.PointLabelTemplate>
      </c1chart:XYDataSeries>
    </c1chart:ChartData>
  </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

提示技巧五：将图表保存为图片

以下方法将一个图表的图片保存为Png文件格式。

Visual Basic

```
Sub Using stm = System.IO.File.Create(fileName)
    c1Chart1.SaveImage(stm, ImageFormat.Png)
End Using
```

C#

```
using (var stm = System.IO.File.Create(fileName))
   {
    c1Chart1.SaveImage(stm, ImageFormat.Png);
   }
```

提示技巧六：打印图表

以下代码在默认的打印机上采用默认的设置打印指定的图表。例如：

Visual Basic

```
Dim pd = New PrintDialog()
  pd.PrintVisual(C1Chart1, "chart")
```

C#

```
new PrintDialog().PrintVisual(c1Chart1, "chart");
```

提示技巧七：混合直角坐标图表类型

您可以容易地在同一个直角坐标系的绘图区通过使用ChartType属性混合显示不同的图表类型。
以下代码创建了三个数据系列，第一个是面积图，第二个是step图，第三个为默认的图表类型（折线图）。

Visual Basic

```
Dim nser As Integer = 3, npts As Integer = 25
  For iser As Integer = 1 To nser
    ' 创建数据数组
    Dim x(npts - 1) As Double, y(npts - 1) As Double
    For ipt As Integer = 0 To npts - 1
      x(ipt) = ipt
      y(ipt) = (1 + 0.05 * iser) * Math.Sin(0.1 * ipt + 0.1 * iser)
    Next
    ' 创建数据系列
    Dim ds = New XYDataSeries()
    ds.XValuesSource = x
    ds.ValuesSource = y
    C1Chart1.Data.Children.Add(ds)
  Next

  '设置图表类型
  C1Chart1.ChartType = ChartType.Line
```

```
  ' 第一系列
  C1Chart1.Data.Children(0).ChartType = ChartType.Area

  ' 第二系列
  C1Chart1.Data.Children(1).ChartType = ChartType.Step
```

C#

```
int nser = 3, npts = 25;
  for (int iser = 0; iser < nser; iser++)
  {
    // 创建数据数组
    double[] x = new double[npts], y = new double[npts];
    for (int ipt = 0; ipt < npts; ipt++)
    {
      x[ipt] = ipt;
      y[ipt] = (1 + 0.05 * iser) * Math.Sin(0.1 * ipt + 0.1 * iser);
    }

    // 创建数据系列
    XYDataSeries ds = new XYDataSeries();
    ds.XValuesSource = x; ds.ValuesSource = y;
    c1Chart1.Data.Children.Add(ds);
  }

  //设置图表类型
  c1Chart1.ChartType = ChartType.Line;

  // 第一系列
  c1Chart1.Data.Children[0].ChartType = ChartType.Area;

  // 第二个系列
  c1Chart1.Data.Children[1].ChartType = ChartType.Step;
```

XAML 快速参考

以下XAML演示如何选择图表类型，选择调色板，设置坐标轴以及为一个折线图添加数据系列：

XAML

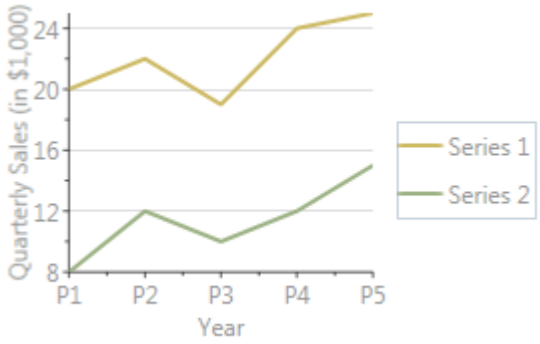
```
<c1:C1Chart x:Name="_chart" Palette="Module" ChartType="Line"
Foreground="#a0000000" Background="#e0ffffff" >

            <c1:C1Chart.View>
                <c1:ChartView>
                    <c1:ChartView.AxisX>
                        <c1:Axis Title="Year" MajorGridStroke="Transparent"
/>
                    </c1:ChartView.AxisX>
                    <c1:ChartView.AxisY>
                        <c1:Axis Title="Quarterly Sales (in $1,000)"
```

```
MajorGridStroke="#40000000" AnnoFormat="n0" />
                    </c1:ChartView.AxisY>
                </c1:ChartView>
            </c1:C1Chart.View>

            <c1:C1ChartLegend Position="Right" />

            <c1:C1Chart.Data>
                <c1:ChartData ItemNames="P1 P2 P3 P4 P5">
                    <c1:DataSeries Label="s1" Values="20, 22, 19, 24, 25"
ConnectionStrokeThickness="6" />
                    <c1:DataSeries Label="s2" Values="8, 12, 10, 12, 15" />
                </c1:ChartData>
            </c1:C1Chart.Data>
        </c1:C1Chart>
```

设置基本的折线图形

以下XAML展示如何声明C1Chart控件，设置ChartType，Theme，以及Palette属性以定义基本的图表外观XAML可以被添加在标签内

下图表由下面的XAML代码生成：



XAML

```
<Window x:Class="WpfApplication1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:c1="http://schemas.componentone.com/xaml/c1chart"
  Title="Window1" Height="300" Width="348" >
  <Grid>
        <!--声明C1Chart控件
      设置，Theme，以及Palette 属性以定义基本的图表外观-->
    <c1:C1Chart
      Name="C1Chart1"
      ChartType="Line"
      Foreground="#a0000000"
      Background="#e0ffffff"
      Theme ="Vista"
```

```
      Palette ="Aspect" >

      <!--定义图表View，包括图表坐标轴。-->
      <c1:C1Chart.View>
        <c1:ChartView>

          <!--定义X-轴（标题，网格）-->
          <c1:ChartView.AxisX>
            <c1:Axis
              Title="Year"
              MajorGridStroke="Transparent"/>
          </c1:ChartView.AxisX>

          <!--定义Y-轴（标题，网格，标注格式）-->
          <c1:ChartView.AxisY>
            <c1:Axis
              Title="Quarterly Sales (in $1,000)"
              MajorGridStroke="#40000000"
              AnnoFormat="n0" />
          </c1:ChartView.AxisY>
        </c1:ChartView>
      </c1:C1Chart.View>

      <!--定义图表数据，包含数据系列。-->
      <c1:C1Chart.Data>
        <c1:ChartData>

          <!-- ItemNames 定义沿着X-轴方向的标签-->
          <c1:ChartData.ItemNames>P1 P2 P3 P4 P5</c1:ChartData.ItemNames>

          <!--每一个DataSeries指定一个标签（显示在图例上），以及系列数据-->
          <c1:DataSeries Label="Series 1" RenderMode="Default" Values="20 22 19 24
25" />
          <c1:DataSeries Label="Series 2" RenderMode="Default" Values="8 12 10 12 15"
/>
        </c1:ChartData>
      </c1:C1Chart.Data>

     <!--添加一个 ChartLegend，停靠到图表的右侧，以便显示一个包含数据系列以及其样式的图例区。-->
      <c1:C1ChartLegend DockPanel.Dock="Right" />
    </c1:C1Chart>
  </Grid>
</Window>
```

设置一个甘特图表

为了创建一个甘特图表，请使用以下XAML标记代码：

XAML

```
<c1chart:C1Chart Margin="0" Name="c1Chart1"
```

```
            xmlns:sys="clr-namespace:System;assembly=mscorlib">
  <c1chart:C1Chart.Resources>
    <x:Array x:Key="start" Type="sys:DateTime" >
      <sys:DateTime>2008-6-1</sys:DateTime>
      <sys:DateTime>2008-6-4</sys:DateTime>
      <sys:DateTime>2008-6-2</sys:DateTime>
    </x:Array>
    <x:Array x:Key="end" Type="sys:DateTime">
      <sys:DateTime>2008-6-10</sys:DateTime>
      <sys:DateTime>2008-6-12</sys:DateTime>
      <sys:DateTime>2008-6-15</sys:DateTime>
    </x:Array>
  </c1chart:C1Chart.Resources>
  <c1chart:C1Chart.Data>
    <c1chart:ChartData>
      <c1chart:ChartData.Renderer>
        <c1chart:Renderer2D Inverted="True" ColorScheme="Point"/>
      </c1chart:ChartData.Renderer>
      <c1chart:ChartData.ItemNames>Task1 Task2 Task3</c1chart:ChartData.ItemNames>
      <c1chart:HighLowSeries HighValuesSource="{StaticResource end}"
                        LowValuesSource="{StaticResource start}"/>
    </c1chart:ChartData>
  </c1chart:C1Chart.Data>
  <c1chart:C1Chart.View>
    <c1chart:ChartView>
      <c1chart:ChartView.AxisX>
        <c1chart:Axis IsTime="True" AnnoFormat="d"/>
      </c1chart:ChartView.AxisX>
    </c1chart:ChartView>
  </c1chart:C1Chart.View>
</c1chart:C1Chart>
```

创建一个堆叠面积图

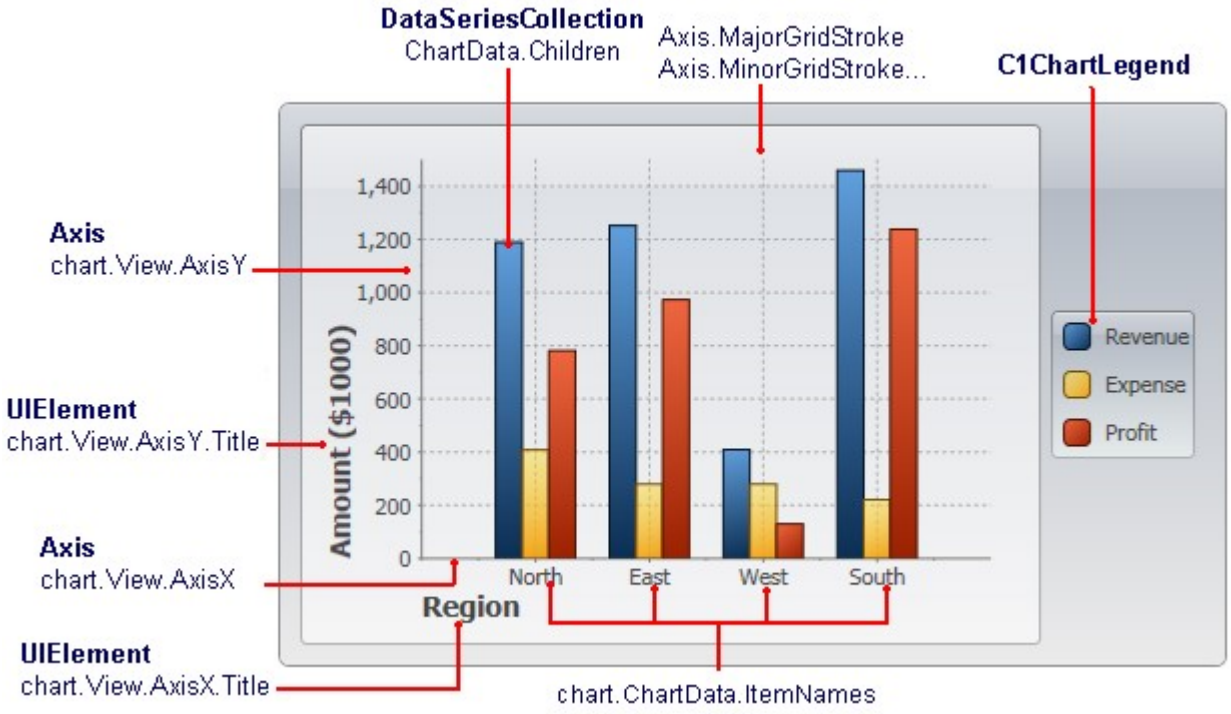
为创建一个堆叠面积图，您应当设置C1Chart.ChartType（在线文档'ChartType 属性'）属性，而不是DataSeries.ChartType（在线文档'ChartType 属性'），如下面的XAML代码所示：

XAML

```
<c1:C1Chart ChartType="AreaStacked" >
        <c1:C1Chart.Data>
          <c1:ChartData ItemNames="P1 P2 P3 P4 P5">
            <c1:DataSeries Label="Series 1" Values="20 22 19 24 25" />
            <c1:DataSeries Label="Series 2" Values="8 12 10 12 15" />
          </c1:ChartData>
        </c1:C1Chart.Data>
        <c1:C1ChartLegend DockPanel.Dock="Right" />
      </c1:C1Chart>
```

概念以及主要属性

为了通过C1Chart（在线文档 'C1Chart 类'）控件创建并格式化一个图表，理解主要的属性到图表元素的映射关系非常有用。下图演示了该映射关系：



参与创建一个典型的图表的步骤如下：

1. 选择图表类型（ChartType（在线文档 'ChartType 属性'）属性）
   C1Chart支持大约三十种图表类型，包括条形图，柱状图，折线图，面积图，饼图，雷达图，极坐标图，蜡烛图以及其他一些图表类型。何种图表类型是最优选择在很大程度上取决于数据本身的特质，这一点我们将在后续章节进行讨论。
2. 设置坐标轴（AxisX以及chart.View.AxisY属性）
   设置坐标轴通常包括制定坐标轴标题，主要和次要间隔的刻度线，显示在刻度线旁边标签的内容以及格式。
3. 添加一个或多个数据系列（chart.Data.Children集合）
   该步骤包括为图表上的每一个系列创建并填充一个DataSeries对象，并添加该对象至chart.Data.Children集合。如果您的数据每一个点金包含一个数值类型的值（Y轴），请使用普通的DataSeries对象。如果您的数据每一个数据点包含两个数值类型的值（X和Y轴），则请使用XYDataSeries对象。
4. 通过Theme以及Palette属性调整图表的外观。
   Theme属性允许您在超过十种以上的内置主题中选择一个，以控制图表的整体外观。Palette属性允许您在超过二十种的内置调色板中选取一个以指定数据系列的颜色。总之，这两个属性组合可以提供多达两百中不同的选项以便毫不费力地创建具有专业外观的图表。

图表类型

该章节将具体介绍C1Chart所有可用的图表类型。

使用内置的类型是设置图表外观的最简单的方式。例如：为设置一个堆叠条形图，为ChartType属性指定相关的字符串：

XAML

```
<c1chart:C1Chart ChartType="BarStacked">
        ...
</c1chart:C1Chart>
```

所有可用的图表类型由ChartType枚举的成员指定。了解更多关于不同图表类型的信息，请单击下方的"另请参见"链接。

面积图

一个面积图将每一个数据系列中的数据点绘制为相连的点，并将该点下方的部分填充。每一个数据系列将该在前一个系列的上方。数据系列可以独立绘制或者堆叠在一起。在WPF版，您还可以创建三维的面积图。WPF及Silverlight版Chart支持一以下面积图类型：

- Area3D
- Area3Dsmoothed
- Area3Dstacked
- Area3Dstacked100pc
- AreaSmoothed
- AreaStacked
- AreaStacked100pc

您可以使用以下标记创建一个面积图：

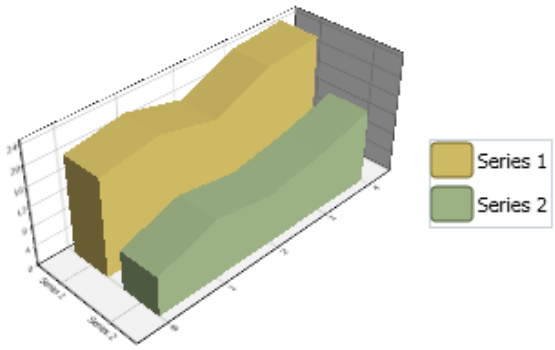
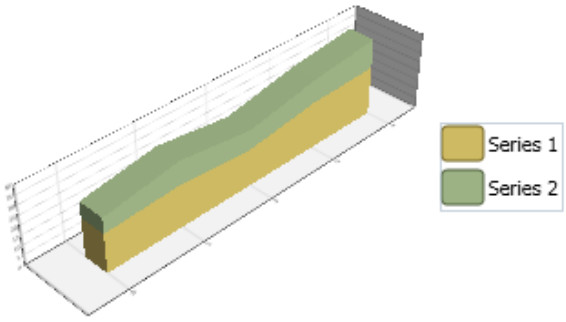
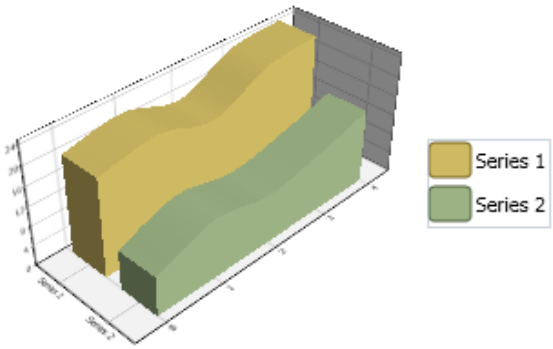
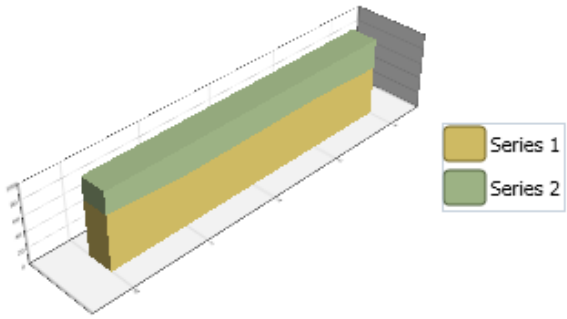
XAML

```
<c1:C1Chart ChartType="Area" >
        <c1:C1Chart.Data>
          <c1:ChartData ItemNames="P1 P2 P3 P4 P5">
            <c1:DataSeries Label="Series 1" Values="20 22 19 24 25" />
            <c1:DataSeries Label="Series 2" Values="8 12 10 12 15" />
          </c1:ChartData>
        </c1:C1Chart.Data>
        <c1:C1ChartLegend DockPanel.Dock="Right" />
      </c1:C1Chart>
```

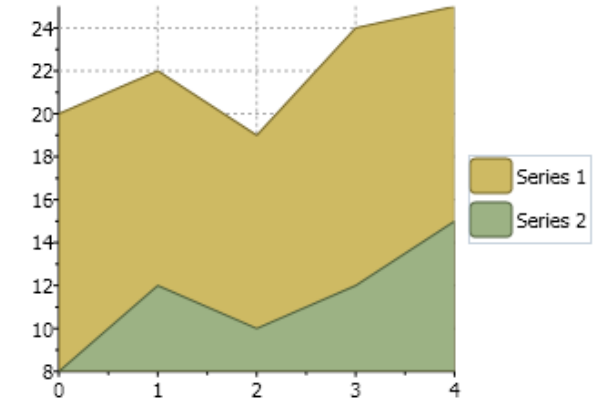
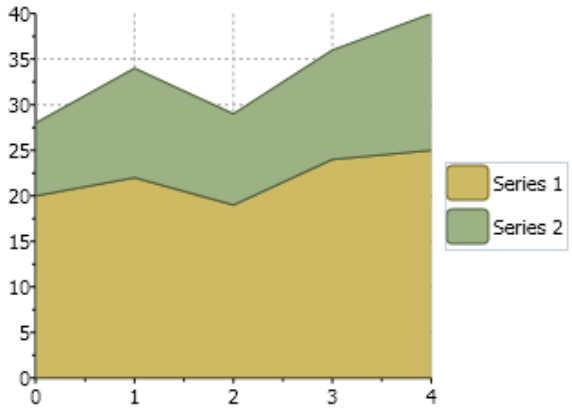
三维面积图（仅WPF版支持）

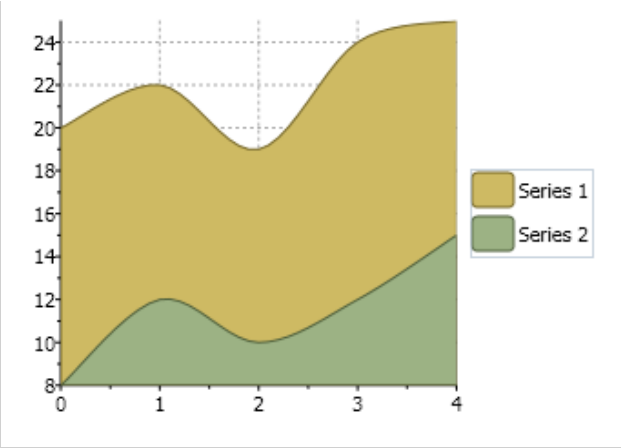
使用AreaShape3D（在线文档 'AreaShape3D 类'）类以访问三维图表绘图元素关联的数据，当鼠标光标悬停在某个绘图区元素上方时获取其值，获取绘图区元素的以像素表示的尺寸大小，指定是否数据点通过平滑曲线连接。以下三维面积图类型在WPF版本中可用：

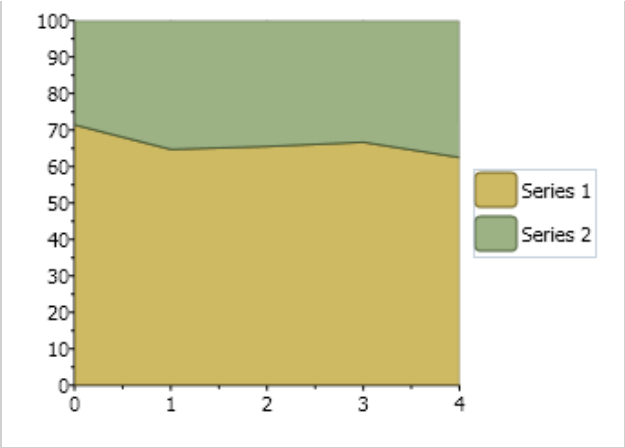
Area3D	Area3Dstacked

	
Area3Dsmoothed	Area3Dstacked100pc
	

标准面积图（WPF及Silverlight）

Area	AreaStacked
	
AreaSmoothed	AreaStacked100pc



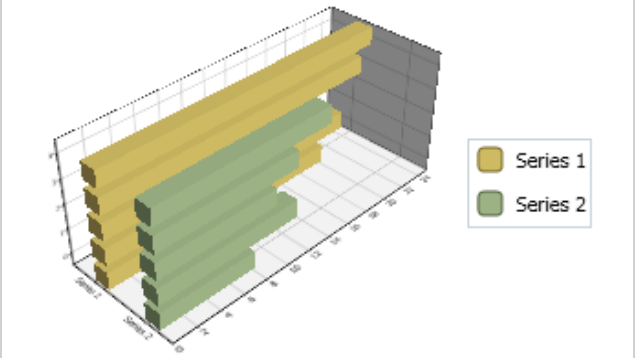
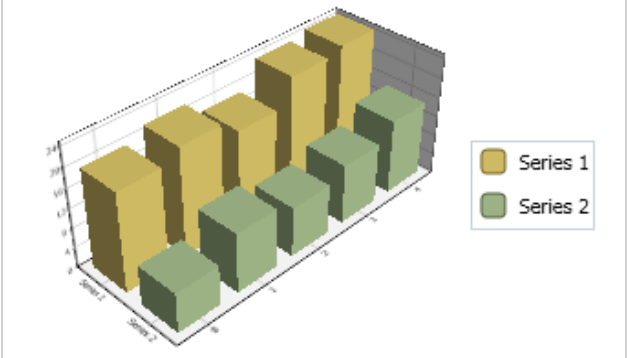
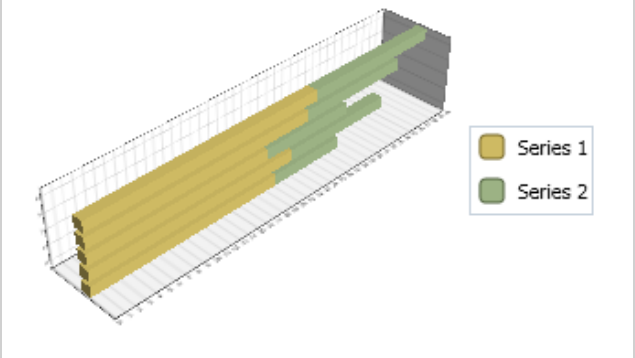
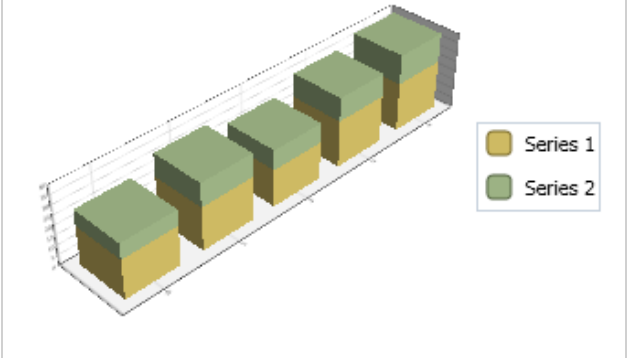


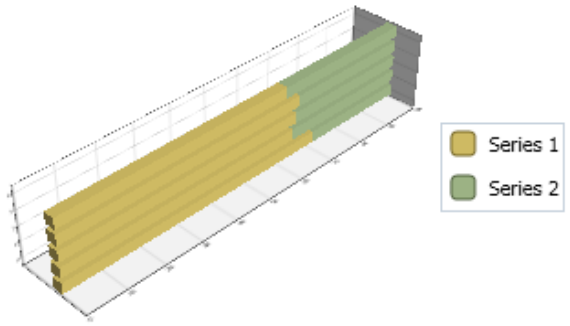
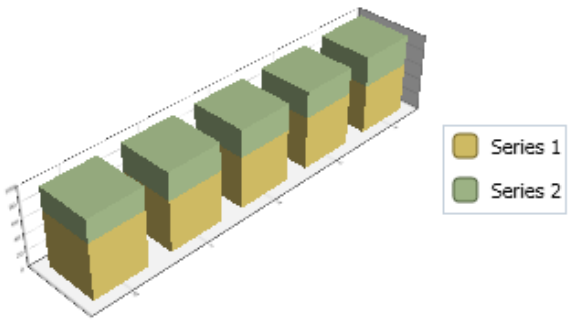
条形图和柱状图

WPF版Chart支持以下条形图类型：

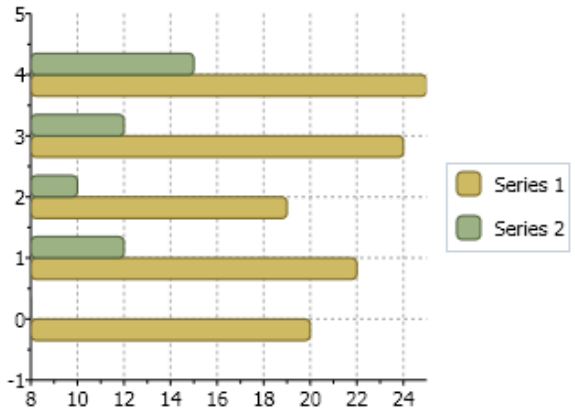
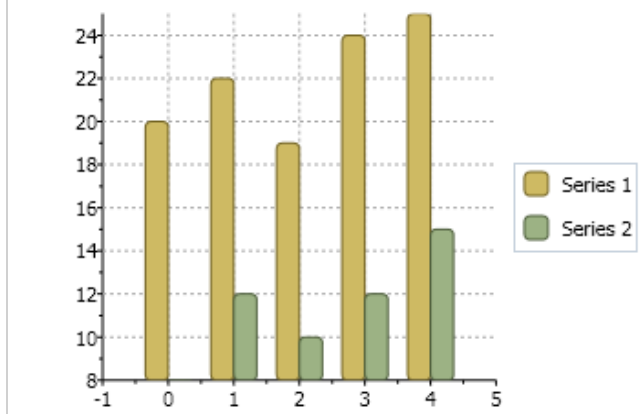
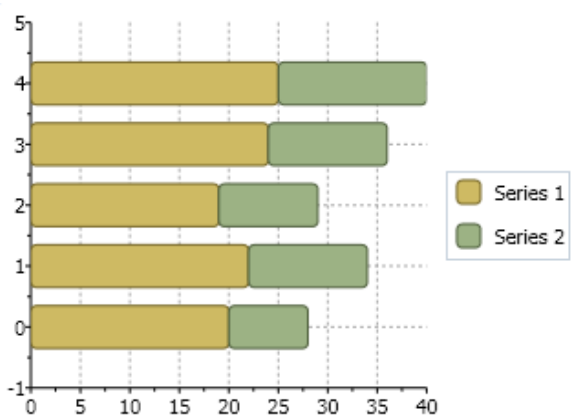
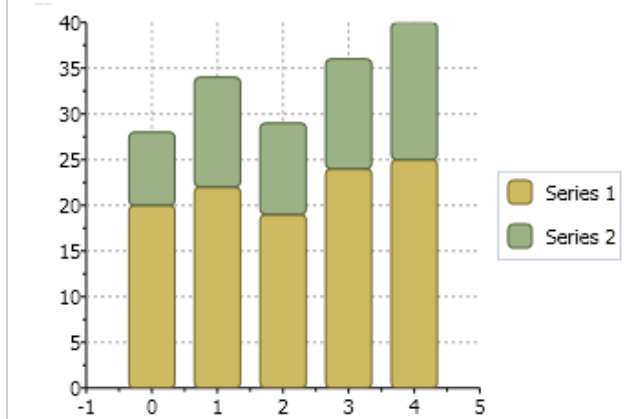
- Bar or Column
- Bar3D or Column3D
- Bar3Dstacked or Column3Dstacked
- Bar3Dstacked100pc or Column3Dstacked100pc
- BarStacked or ColumnStacked
- BarStacked100pc of ColumnStacked100pc

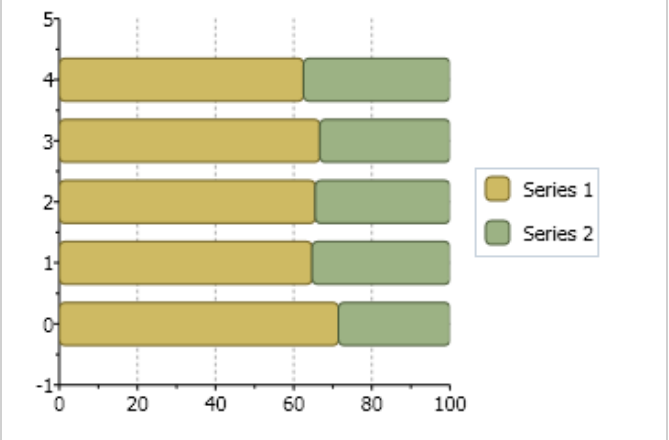
三维条形图和柱形图（仅WPF）

3D条形图和柱形图	
	
3DBarStacked和3DColumnStacked Charts	
	

Bar3DStacked100pc和Column3DStacked100pc Charts	
	

条形图和柱状图

条形图和柱状图	
	
BarStacked和ColumnStacked Charts	
	
BarStacked100pc和ColumnStacked100pc Charts	



改变柱状图以及条形图为圆角

默认情况下，条形图和柱状图不显示圆角。矩形框圆角的半径可以通过Bar 类型进行设置，例如：

C#

```
ds.Symbol = new Bar() { RadiusX=5, RadiusY=5};
```

为一个柱状图创建一个鼠标单击事件

当单击任意柱状图中任意柱形时，您可以通过MouseDown以及MouseLeave事件添加动画，如下XAML标记代码所示：

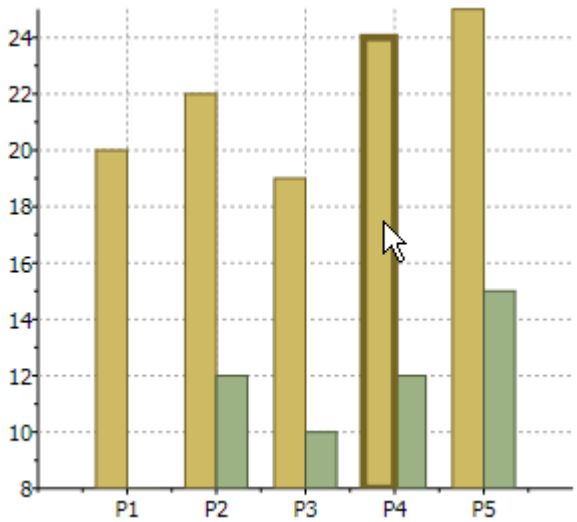
XAML

```
<Window x:Class="MouseEvent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          xmlns:sys="clr-namespace:System;assembly=mscorlib"
    Title="Window1" Height="300" Width="300"
xmlns:c1chart="http://schemas.componentone.com/xaml/c1chart" Loaded="Window_Loaded">
    <Grid>
        <Grid.Resources>
            <Style x:Key="sstyle" TargetType="{x:Type c1chart:PlotElement}">
                <Setter Property="StrokeThickness" Value="1" />
                <Setter Property="Canvas.ZIndex" Value="0" />
                <Style.Triggers>
                    <EventTrigger RoutedEvent="c1chart:PlotElement.MouseDown">
                        <BeginStoryboard>
                            <Storyboard>
                                <Int32Animation Storyboard.TargetProperty="
(Panel.ZIndex)"
                             To="1" />
                                <DoubleAnimation
Storyboard.TargetProperty="StrokeThickness"
                            To="4" Duration="0:0:0.3"
                              AutoReverse="True"
```

```
                            RepeatBehavior="Forever" />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
                <EventTrigger RoutedEvent="c1chart:PlotElement.MouseLeave">
                    <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimation
Storyboard.TargetProperty="StrokeThickness" />
                            <Int32Animation Storyboard.TargetProperty="
(Panel.ZIndex)" />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Style.Triggers>
        </Style>
    </Grid.Resources>
    <c1chart:C1Chart Margin="0" Name="c1Chart1" ChartType="Column">
        <c1chart:C1Chart.Data>
            <c1chart:ChartData>
                <c1chart:ChartData.ItemNames>P1 P2 P3 P4
P5</c1chart:ChartData.ItemNames>
                <c1chart:DataSeries SymbolStyle="{StaticResource sstyle}"
Values="20
22 19 24 25" />
                <c1chart:DataSeries SymbolStyle="{StaticResource sstyle}"
Values="8
12 10 12 15" />
            </c1chart:ChartData>
        </c1chart:C1Chart.Data>
    </c1chart:C1Chart>
</Grid>
</Window>
```

该主题演示以下内容：

单击任意柱形，可以注意到矩形框周围的动画效果：



指定数据系列每一个条形或柱形的颜色

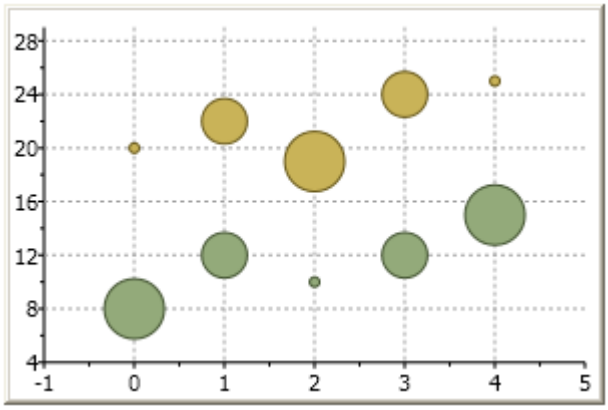
您可以在数据系列的PlotElementLoaded（在线文档 'PlotElementLoaded 事件'）事件中指定每一个条形或柱形的颜色，通过以下代码：

C#

```
var palette = new Brush[] { Brushes.Red, Brushes.Plum, Brushes.Purple };
      dataSeries.PlotElementLoaded += (s, e) =>
        {
          PlotElement pe = (PlotElement)s;
          if (pe.DataPoint.PointIndex >= 0)
            pe.Fill = palette[pe.DataPoint.PointIndex % palette.Length];
        };
```

气泡图

下图显示当您设置ChartType属性的值为Bubble时，该气泡图的外观：



下面的XAML标记语言将创建一个气泡图：

XAML

```
<c1chart:C1Chart ChartType="Bubble"
       c1chart:BubbleOptions.MinSize="5,5"
       c1chart:BubbleOptions.MaxSize="30,30"
       c1chart:BubbleOptions.Scale="Area">
    <c1chart:C1Chart.Data>
      <c1chart:ChartData>
        <c1chart:BubbleSeries Values="20 22 19 24 25" SizeValues="1 2 3 2 1" />
        <c1chart:BubbleSeries Values="8 12 10 12 15" SizeValues="3 2 1 2 3"/>
      </c1chart:ChartData>
    </c1chart:C1Chart.Data>
  </c1chart:C1Chart>
```

K线图

K线图表示一种特殊的Hi-Lo-Open-Close图表，用于显示开闭值之间的关系以及最高值和最低值。和Hi-Lo-Open-Close图表类似，K线图使用相同的价格数据（时间，最高值，最低值，开盘价和收盘价），唯一不同的是K线图包含一个粗粗的K线一样的身体。

K线图由以下元素组成：

- K线

  粗粗的像K线一样的身体使用颜色和尺寸以揭示关于开盘价和收盘价之间关系的额外信息。

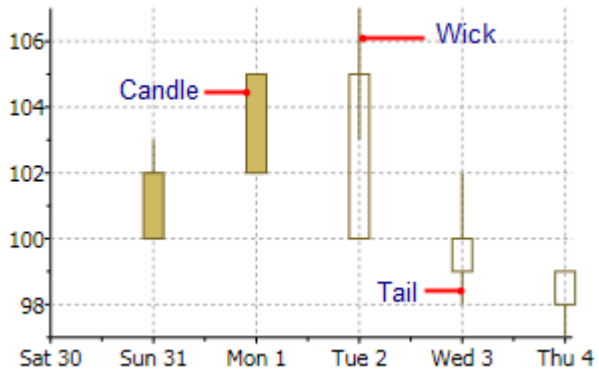
  - 一根长长的透明的K线显示了购买压力，而一个较短的，填充有颜色的K线则显示了销售压力。
  - 一根空心的或透明的K线，预示着股票价格的上涨（高于开盘价）。在一根空心的K线形状中，K线体的底部表示开盘价，而K线体的顶部则表示收盘价。
  - 一根填充了颜色的K线则表示了一个正在下跌的股票价格（开盘价高于收盘价）。在一个填充的K线图形中，K线体的顶端代表着开盘价，而K线体的底部则代表收盘价。

- 烛芯

  烛芯是K线上方的线，描述的是高价的范围。

- 尾线
  尾线是K线底部的线，描述了低价的范围。

下图通过标签描述了每一个K线图表中的元素：



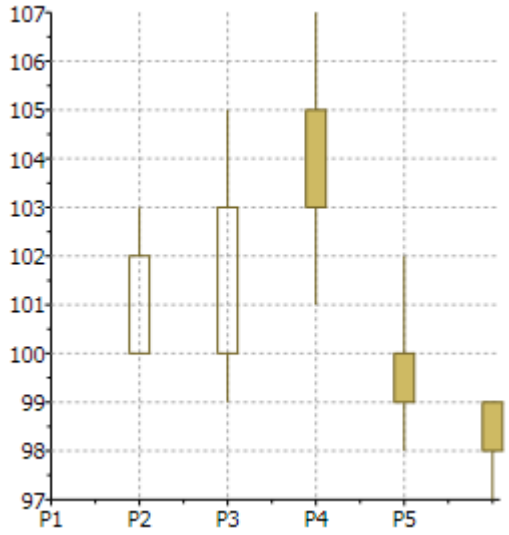
下面的图像表示当您设置了ChartType属性的值为Candle，并指定了XValues（在线文档 'XValues 属性'），HighValues（在线文档 'HighValues 属性'），LowValues（在线文档 'LowValues 属性'），OpenValues（在线文档 'OpenValues 属性'），以及CloseValues（在线文档 'CloseValues 属性'）的值时K线图的外观，如下所示：

XAML

```
<c1chart:C1Chart ChartType="Candle">
    <c1chart:C1Chart.Data>
      <c1chart:ChartData>
        <c1chart:HighLowOpenCloseSeries
            XValues="1 2 3 4 5"
           HighValues="103 105 107 102 99"
           LowValues="100 99 101 98 97"
           OpenValues="100 100 105 100 99"
           CloseValues="102 103 103 99 98"
```

```
        />
      </c1chart:ChartData>
    </c1chart:C1Chart.Data>
  </c1chart:C1Chart>
```

以上XAML标记代码将生成一个类似于以下图片的图表：



改变蜡烛的宽度

为改变蜡烛的宽度，请使用SymbolSize（在线文档 'SymbolSize 属性'）属性如下：

C#

```
ds.SymbolSize = new Size(5, 5);
```

HighLowOpenClose 图表

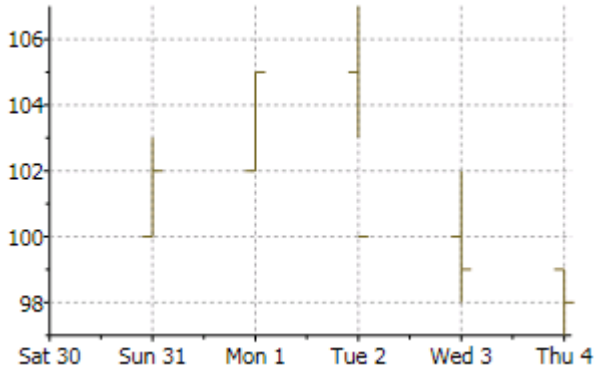
常见的图表类型和金融图表类型之间的区别在于，HighLowOpenClose 图表需要一类特殊的数据系列对象类型，HighLowOpenCloseSeries （在线文档 'HighLowOpenCloseSeries 类'）。在这种类型的数据系列中，每一点对应一段时间（通常是一天），并包含五个值：

- 时间
- 阶段初期的价格（开盘）
- 时间段结束时的价格（收盘）
- 在期间内的最低价格（低价）
- 在期间内的最高价格（高价）
- 创建金融图表，您需要提供所有这些值。

下面的图像表示当您设置了ChartType属性的值为HighLowOpenClose，并指定了XValues（在线文档 'XValues 属性'），HighValues（在线文档 'HighValues 属性'），LowValues（在线文档 'LowValues 属性'），OpenValues（在线文档 'OpenValues 属性'），以及CloseValues（在线文档 'CloseValues 属性'）的值时HighLowOpenClose图表的外观，如下所示：

XAML

```
<c1chart:C1Chart ChartType="HighLowOpenClose">
    <c1chart:C1Chart.Data>
      <c1chart:ChartData>
        <c1chart:HighLowOpenCloseSeries
            XValues="1 2 3 4 5"
           HighValues="103 105 107 102 99"
           LowValues="100 99 101 98 97"
           OpenValues="100 100 105 100 99"
           CloseValues="102 103 103 99 98"
         />
      </c1chart:ChartData>
    </c1chart:C1Chart.Data>
  </c1chart:C1Chart>
```



在代码中创建一个Hi-Lo-Open-Close图表

以编程方式创建一个HiLowOpenClose 图表，通过以下代码：

C#

```
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries()
{
XValueBinding = new System.Windows.Data.Binding("NumberOfDay"),
HighValueBinding = new System.Windows.Data.Binding("High"),
LowValueBinding = new System.Windows.Data.Binding("Low"),
OpenValueBinding = new System.Windows.Data.Binding("Open"),
CloseValueBinding = new System.Windows.Data.Binding("Close"),
SymbolStrokeThickness = 1,     SymbolSize = new Size(5, 5)
}
ds.PlotElementLoaded += (s, e) =>
{
PlotElement pe = (PlotElement)s;
double open = (double)pe.DataPoint["OpenValues"];
double close = (double)pe.DataPoint["CloseValues"];
if (open > close)
{
pe.Fill = green;
pe.Stroke = green;
```

```
}
else
{
pe.Fill = red;
pe.Stroke = red;
}
};
```

例如，如果值由应用程序以集合的方式提供，则您可以使用以下代码创建数据系列：

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();
ds.XValuesSource = dates; // 数值沿着x轴方向
ds.OpenValuesSource = open;
ds.CloseValuesSource = close;
ds.HighValuesSource = hi;
ds.LowValuesSource = lo;

// 向图表添加数据系列
chart.Data.Children.Add(ds);

// 设置图表类型
chart.ChartType = isCandle
  ? ChartType.Candle
  : ChartType.HighLowOpenClose;
```

另一种选择是使用数据绑定。例如，如果可用的数据为StockQuote对象的集合，比如说：

C#

```
public class Quote
{
  public DateTime Date { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
}
```

Then the code that creates the data series would be as follows:

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();

// 绑定全部的五个值
ds.XValueBinding = new Binding("Date"); // 数值沿着x轴方向
ds.OpenValueBinding = new Binding("Open");
ds.CloseValueBinding = new Binding("Close");
ds.HighValueBinding = new Binding("High");
```

```
ds.LowValueBinding = new Binding("Low");

// 向图表添加数据系列
chart.Data.Children.Add(ds);

// 设置图表类型
chart.ChartType = isCandle
  ? ChartType.Candle
  : ChartType.HighLowOpenClose;
```

在代码中创建一个Hi-Lo-Open-Close图表

以编程方式创建一个HiLowOpenClose 图表，通过以下代码：

C#

```
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries()
{
XValueBinding = new System.Windows.Data.Binding("NumberOfDay"),
HighValueBinding = new System.Windows.Data.Binding("High"),
LowValueBinding = new System.Windows.Data.Binding("Low"),
OpenValueBinding = new System.Windows.Data.Binding("Open"),
CloseValueBinding = new System.Windows.Data.Binding("Close"),
SymbolStrokeThickness = 1,     SymbolSize = new Size(5, 5)
}
ds.PlotElementLoaded += (s, e) =>
{
PlotElement pe = (PlotElement)s;
double open = (double)pe.DataPoint["OpenValues"];
double close = (double)pe.DataPoint["CloseValues"];
if (open > close)
{
pe.Fill = green;
pe.Stroke = green;
}
else
{
pe.Fill = red;
pe.Stroke = red;
}
};
```

例如，如果值由应用程序以集合的方式提供，则您可以使用以下代码创建数据系列：

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();
ds.XValuesSource = dates; // 数值沿着x轴方向
```

```
ds.OpenValuesSource = open;
ds.CloseValuesSource = close;
ds.HighValuesSource = hi;
ds.LowValuesSource = lo;

// 向图表添加数据系列
chart.Data.Children.Add(ds);

// 设置图表类型
chart.ChartType = isCandle
  ? ChartType.Candle
  : ChartType.HighLowOpenClose;
```

另一种选择是使用数据绑定。例如，如果可用的数据为StockQuote对象的集合，比如说：

C#

```
public class Quote
{
  public DateTime Date { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
}
```

Then the code that creates the data series would be as follows:

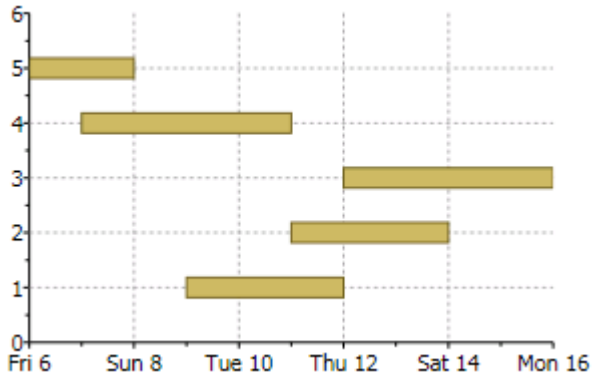
C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();

// 绑定全部的五个值
ds.XValueBinding = new Binding("Date"); // 数值沿着x轴方向
ds.OpenValueBinding = new Binding("Open");
ds.CloseValueBinding = new Binding("Close");
ds.HighValueBinding = new Binding("High");
ds.LowValueBinding = new Binding("Low");

// 向图表添加数据系列
chart.Data.Children.Add(ds);

// 设置图表类型
chart.ChartType = isCandle
  ? ChartType.Candle
  : ChartType.HighLowOpenClose;
```

甘特图

甘特图使用类型为HighLowSeries的对象作为数据系列（在线文档 'HighLowSeries 类'）。每一个数据系列表示一个单独的任务，每一个任务都有一个起始值和结束值的集合。简单的任务有一个开始值和一个结束值。由多个顺序的子任务组成的任务有多个开始和结束值对。

下图表示一个甘特图：



为了演示甘特图，让我们从定义一个Task对象开始：

C#

```
class Task
{
  public string Name { get; set; }
  public DateTime Start { get; set; }
  public DateTime End { get; set; }
  public bool IsGroup { get; set; }
  public Task(string name, DateTime start, DateTime end, bool isGroup)
  {
    Name = name;
    Start = start;
    End = end;
    IsGroup = isGroup;
  }
}
```

接下来，我们定义一个方法，创建一组将显示为甘特图的任务对象：

C#

```
Task[] GetTasks()
{
  return new Task[]
  {
    new Task("Alpha", new DateTime(2008,1,1), new DateTime(2008,2,15), true),
    new Task("Spec", new DateTime(2008,1,1), new DateTime(2008,1,15), false),
    new Task("Prototype", new DateTime(2008,1,15), new DateTime(2008,1,31), false),
    new Task("Document", new DateTime(2008,2,1), new DateTime(2008,2,10), false),
    new Task("Test", new DateTime(2008,2,1), new DateTime(2008,2,12), false),
    new Task("Setup", new DateTime(2008,2,12), new DateTime(2008,2,15), false),
```

```
    new Task("Beta", new DateTime(2008,2,15), new DateTime(2008,3,15), true),
    new Task("WebPage", new DateTime(2008,2,15), new DateTime(2008,2,28), false),
    new Task("Save bugs", new DateTime(2008,2,28), new DateTime(2008,3,10), false),
    new Task("Fix bugs", new DateTime(2008,3,1), new DateTime(2008,3,15), false),
    new Task("Ship", new DateTime(2008,3,14), new DateTime(2008,3,15), false),
  };
}
```

现在这个任务已经建立，我们已经准备好创建甘特图：

C#

```
private void CreateGanttChart()
{
  // 清除当前图表
  c1Chart.Reset(true);

  // 设置图表类型
  c1Chart.ChartType = ChartType.Gantt;

  // 生成图表
  var tasks = GetTasks();
  foreach (var task in tasks)
  {
    // 为每一个任务创建一个数据系列
    var ds = new HighLowSeries();
    ds.Label = task.Name;
    ds.LowValuesSource = new DateTime[] { task.Start };
    ds.HighValuesSource = new DateTime[] { task.End };
    ds.SymbolSize = new Size(0, task.IsGroup ? 30 : 10);

    // 添加数据系列至图表
    c1Chart.Data.Children.Add(ds);
  }

  // 沿着Y轴显示任务名称
  c1Chart.Data.ItemNames =
    (from task in tasks select task.Name).ToArray();

  // 自定义Y轴
  var ax = c1Chart.View.AxisY;
  ax.Reversed = true;
  ax.MajorGridStroke = null;

  // 自定义X轴
  ax = c1Chart.View.AxisX;
  ax.MajorGridStrokeDashes = null;
  ax.MajorGridFill = new SolidColorBrush(Color.FromArgb(20, 120, 120, 120));
  ax.Min = new DateTime(2008, 1, 1).ToOADate();
}
```

在清除完C1Chart并设置了图表类型之后，该代码枚举每一个任务并为每一个任务创建了一个HighLowSeries对象。

除了设置系列标签，LowValuesSource以及HighValuesSource 属性之外，该代码通过SymbolSize属性以设置每一个任务条形的高度。在这个示例中，我们将一些任务定义为"组"任务，并使它们比一般任务更高。

接下来，我们使用LINQ语句来提取任务名称并将其设置给ItemNames属性。这将使得C1Chart沿Y轴显示任务名称。

最后，该代码还对坐标轴进行自定义。Y轴被设置为反向显示，因此第一个任务显示在图表的顶部。坐标轴同时被设置显示垂直的网格线和交替色的条带。

在标记语言中创建甘特图

为创建甘特图，使用下面的XAML标记：

XAML

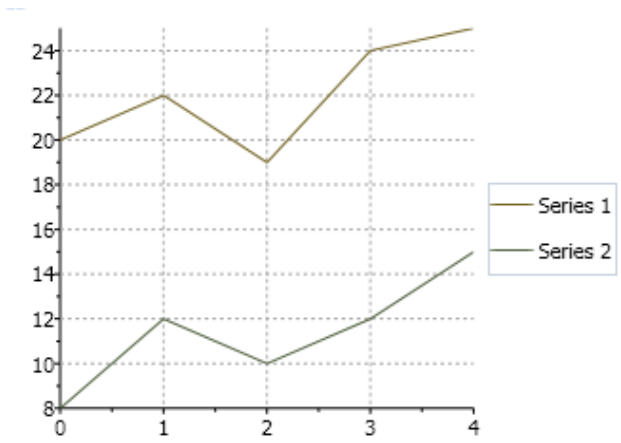
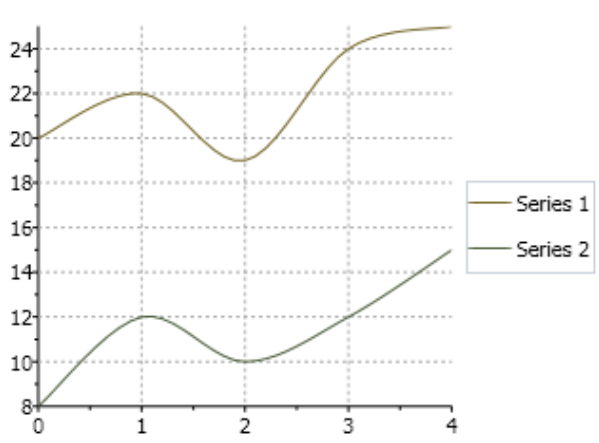
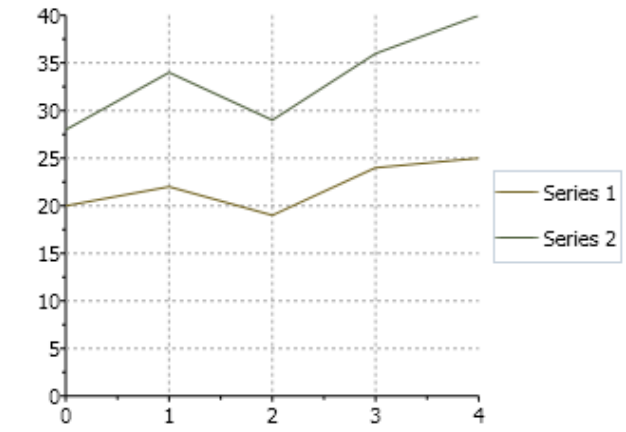
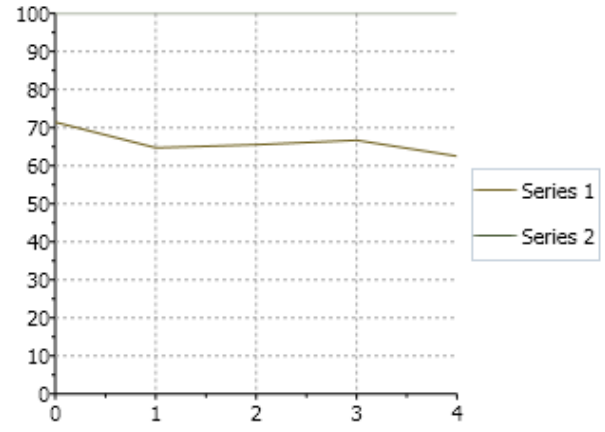
```
<c1chart:C1Chart Margin="0" Name="c1Chart1"
            xmlns:sys="clr-namespace:System;assembly=mscorlib">
  <c1chart:C1Chart.Resources>
    <x:Array x:Key="start" Type="sys:DateTime" >
      <sys:DateTime>2008-6-1</sys:DateTime>
      <sys:DateTime>2008-6-4</sys:DateTime>
      <sys:DateTime>2008-6-2</sys:DateTime>
    </x:Array>
    <x:Array x:Key="end" Type="sys:DateTime">
      <sys:DateTime>2008-6-10</sys:DateTime>
      <sys:DateTime>2008-6-12</sys:DateTime>
      <sys:DateTime>2008-6-15</sys:DateTime>
    </x:Array>
  </c1chart:C1Chart.Resources>
  <c1chart:C1Chart.Data>
    <c1chart:ChartData>
      <c1chart:ChartData.Renderer>
        <c1chart:Renderer2D Inverted="True" ColorScheme="Point"/>
      </c1chart:ChartData.Renderer>
      <c1chart:ChartData.ItemNames>Task1 Task2 Task3</c1chart:ChartData.ItemNames>
      <c1chart:HighLowSeries HighValuesSource="{StaticResource end}"
                        LowValuesSource="{StaticResource start}"/>
    </c1chart:ChartData>
  </c1chart:C1Chart.Data>
  <c1chart:C1Chart.View>
    <c1chart:ChartView>
      <c1chart:ChartView.AxisX>
        <c1chart:Axis IsTime="True" AnnoFormat="d"/>
      </c1chart:ChartView.AxisX>
    </c1chart:ChartView>
  </c1chart:C1Chart.View>
</c1chart:C1Chart>
```

折线图

WPF及Silverlight版Chart支持以下类型的折线图：

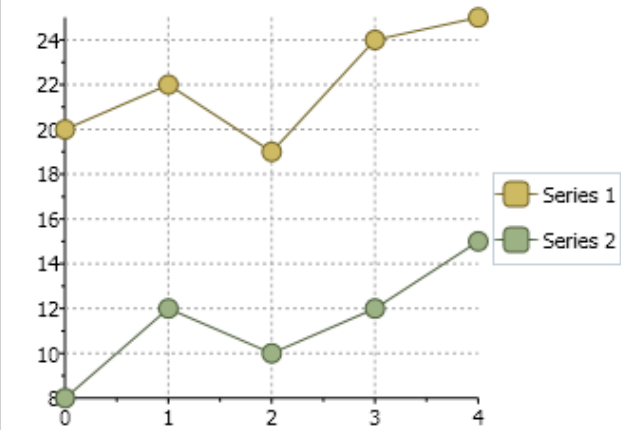
- Line
- LineSmoothed
- LineStacked
- LineStacked100pc
- LineSymbols
- LineSymbolsSmoothed
- LineSymbolsStacked
- LineSymbolsStacked100pc

常规折线图

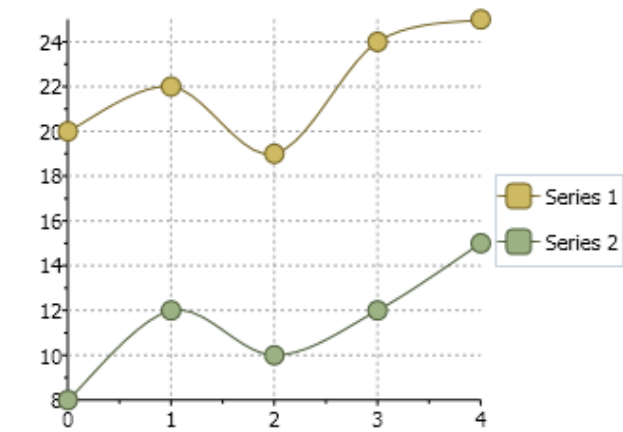
Line	LineSmoothed
	
LineStacked	LineStacked100pc
	

LineSymbol Charts

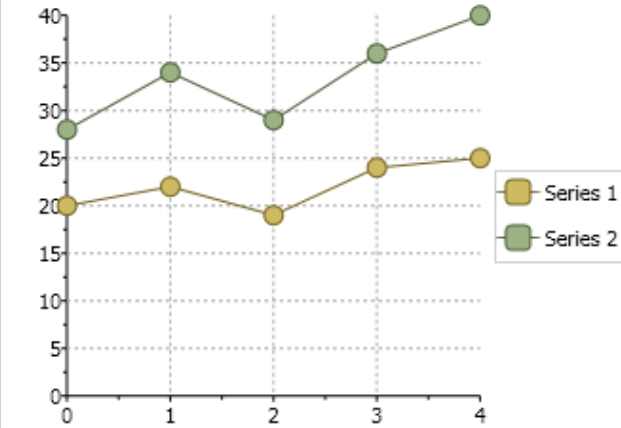
LineSymbols	LineSymbolsSmoothed

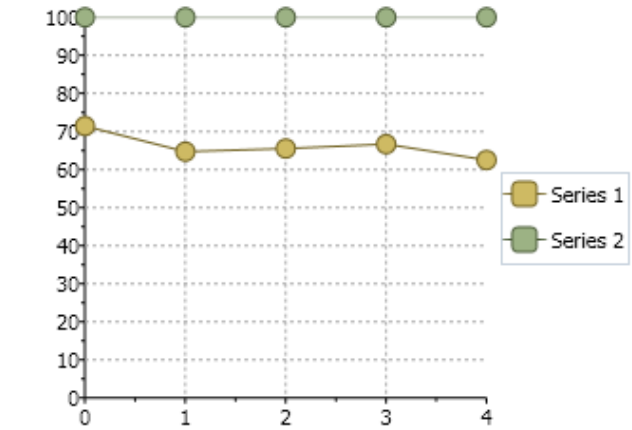


LineSymbolsStacked



LineSymbolsStacked100pc





饼图

饼图通常用于显示简单的值。它们具有视觉上的吸引力，而且经常会显示三维效果，如阴影和旋转。

和C1Chart中其他的图表类型相比，饼图所具有的一个显著不同，每一个系列代表饼图中的一块。因此，你永远不会有一个单一系列的饼图（他们将是一个整圆）。在大多数情况下，饼图有多个系列（每片一个），在每个系列中仅具有一个数据点。C1Chart将具有多个数据点的系列在图表中展示为多个饼的形状。

如果您想通过XAML标记创建一个饼图，则标记语言应当类似以下的代码：

XAML

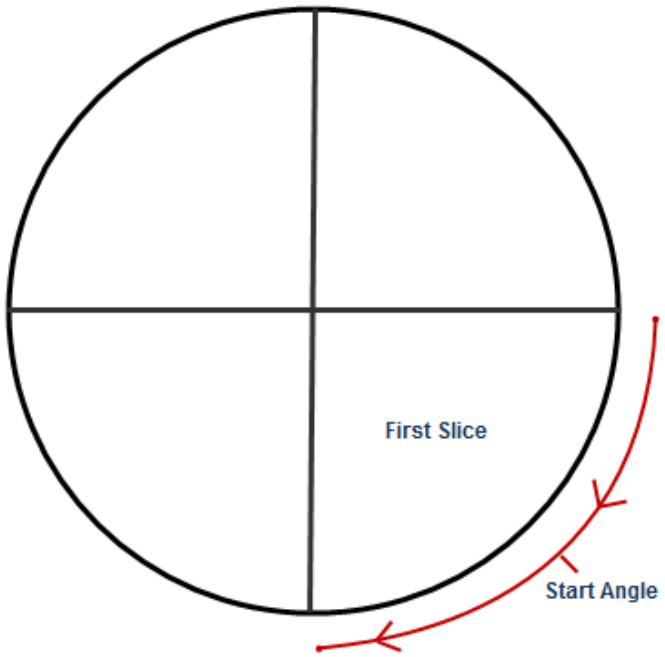
```
<c1chart:C1Chart Name="c1Chart1" ChartType="Pie">
    <c1chart:C1Chart.Data>
      <c1chart:ChartData>
       <c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>
          <c1chart:DataSeries Values="20 22 19 24 25" />
       </c1chart:ChartData>
     </c1chart:C1Chart.Data>
   <c1chart:C1ChartLegend DockPanel.Dock="Right" />
</c1chart:C1Chart>
```

有一些特殊的属性，可以帮助您自定义您的饼图控件。通过使用SetStartingAngle或Direction属性，您可以改变饼图显示的起始角度

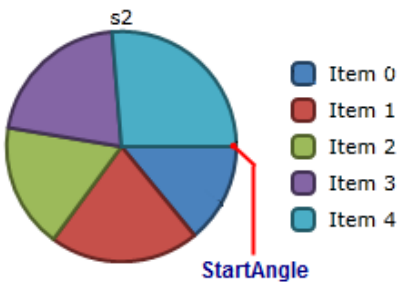
或者改变切片显示的方向。您同时也可以通过BasePieRenderer 属性将一个饼的切片从主图表分离显示，以高亮显示该值。

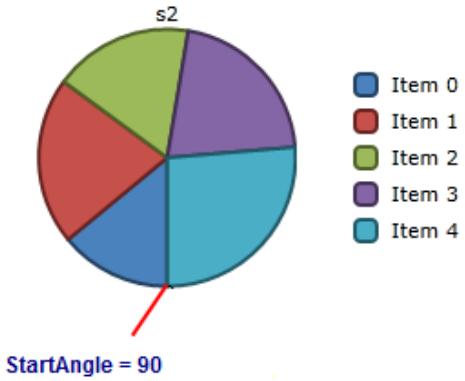
起始角

PieOptions.SetStartingAngle属性定义饼图中的一个切片的位置。饼图第一片的位置始终从90度位置开始显示。启示角从90度角位置顺时针方向计算。



使用PieOptions.SetStartingAngle属性指定第一个系列的切片显示的位置。您可以在下图中看到当设置起始角为90度时会发生什么：





Direction

使用 PieOptions.Direction 附加属性指定图表绘制椭圆弧的方向。选项包括：

- 逆时针：指定在逆时针方向（负角）方向绘制圆弧。
- 顺时针：指定以顺时针方向（正角度）绘制圆弧。

SeriesLabelTemplate

使用PieOptions.SeriesLabelTemplate属性指定一个饼图的系列标签模板。创建一个DataTemplate 以设置系列标签的内容。

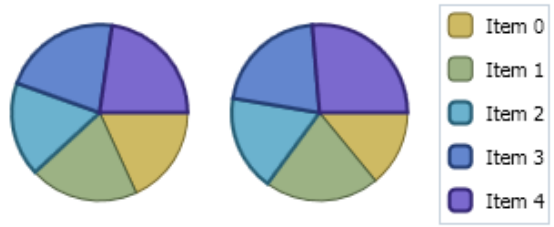
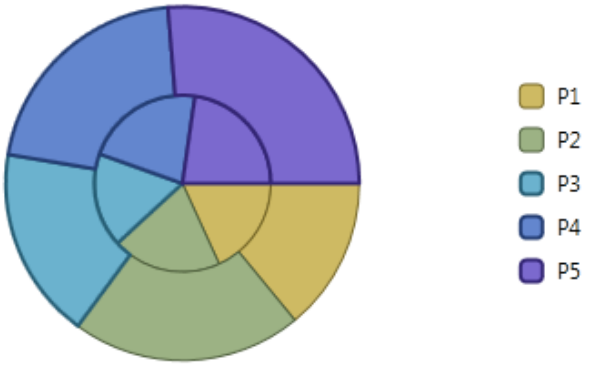
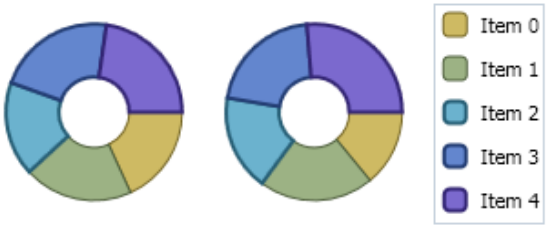
分离显示饼图

一个饼图的切片可以通过爆炸分离进行强调，从余下的饼图切片中突出显示。使用系列的Offset属性以设置分离显示的切片距离饼图中心的偏移量。该偏移量以饼图半径的百分比表示。

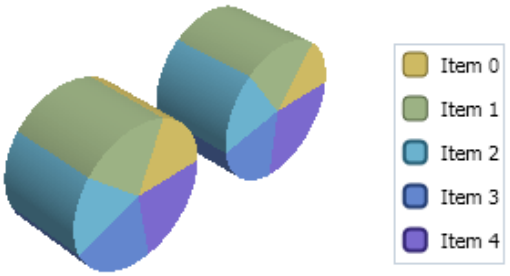
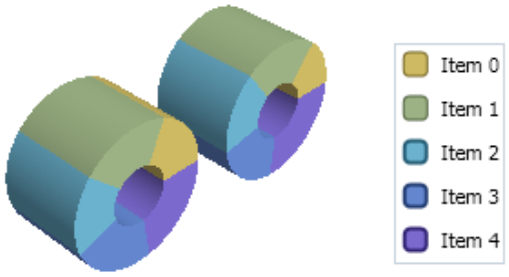
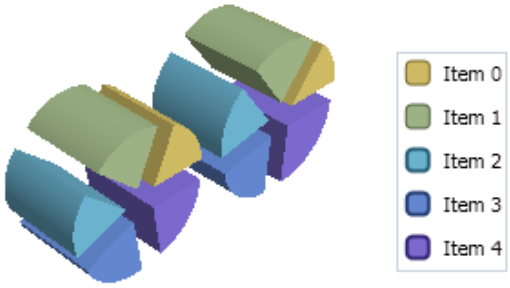
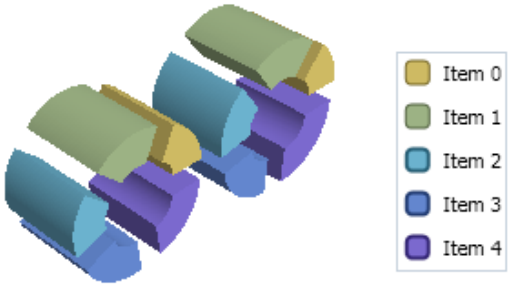
WPF及Silverlight版Chart支持以下类型的饼图：

- Pie
- Pie Stacked
- 3D Pie
- 3D Doughnut Pie
- 3D Exploded Pie
- 3D Exploded Doughnut Pie
- Doughnut Pie
- Exploded Pie
- Exploded Doughnut Pie

常规饼图 （WPF 及 Silverlight）

Pie	PieStacked
	
DoughnutPie	ExplodedPie
	
ExplodedDoughnutPie	
	

3D Pie Charts (仅 WPF)

3DPie	3DDoughnutPie
	
3DExplodedPie	3DExplodedDoughnutPie
	

增加连接线，以防止饼重叠

您可以通过PlotElement.LabelLine 附加属性添加连接线，如以下XAML代码所示：

XAML

```
<c1:DataSeries.PointLabelTemplate>
  <DataTemplate>
    <Border BorderBrush="DarkGray" BorderThickness="1" Background="LightGray"
            c1:PlotElement.LabelAlignment="Auto"
            c1:PlotElement.LabelOffset="30,0">
      <TextBlock Text="{Binding Value, StringFormat=0}" />
      <c1:PlotElement.LabelLine>
        <Line Stroke="LightGray" StrokeThickness="2" />
      </c1:PlotElement.LabelLine>
    </Border>
  </DataTemplate>
</c1:DataSeries.PointLabelTemplate>
```

添加标签到饼图

要将多个值添加到饼图标签中，可以创建如下标签模板：

XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="Pie">
      <c1chart:C1Chart.Resources>
        <DataTemplate x:Key="lbl">
          <StackPanel>
            <StackPanel Orientation="Horizontal">
              <TextBlock Text="{Binding Path=Name}" />
              <TextBlock Text="=" />
              <TextBlock Text="{Binding Path=Value}" />
            </StackPanel>
            <TextBlock Text="{Binding Path=PercentageSeries,Converter={x:Static
c1chart:Converters.Format}, ConverterParameter=#.#%}"/>
          </StackPanel>
        </DataTemplate>
      </c1chart:C1Chart.Resources>
      <c1chart:C1Chart.Data>
        <c1chart:ChartData>
          <c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>
          <c1chart:DataSeries Values="20 22 19 24 25" PointLabelTemplate="
{StaticResource lbl}" />
        </c1chart:ChartData>
      </c1chart:C1Chart.Data>
      <c1chart:C1ChartLegend DockPanel.Dock="Right" />
    </c1chart:C1Chart>
```

改变所有切片的偏移量

要更改所有饼图切片的偏移量，使用以下代码

C#

```
chart.DataContext = new double[] { 1, 2, 3 };
      chart.ChartType = ChartType.Pie;
      chart.Loaded += (s, e) => ((BasePieRenderer)chart.Data.Renderer).Offset = 0.1;
```

设置三维饼图的默认视角

要设置"三维饼图"的默认视角，使用以下代码：

C#

```
chart.View.Camera.Transform = new RotateTransform3D(new AxisAngleRotation3D(new
Vector3D(0,0,1),45));
```

极坐标图表

极坐标图表将每一个系列作为（θ，r）的值在X和Y轴上进行绘制。

- θ-从图表起始位置的旋转量。θ可以以角度指定（默认值），或者以弧度表示。r-表示到图表原点的距离

- 因为X轴是一个圆，X轴的最大值和最小值是固定的。

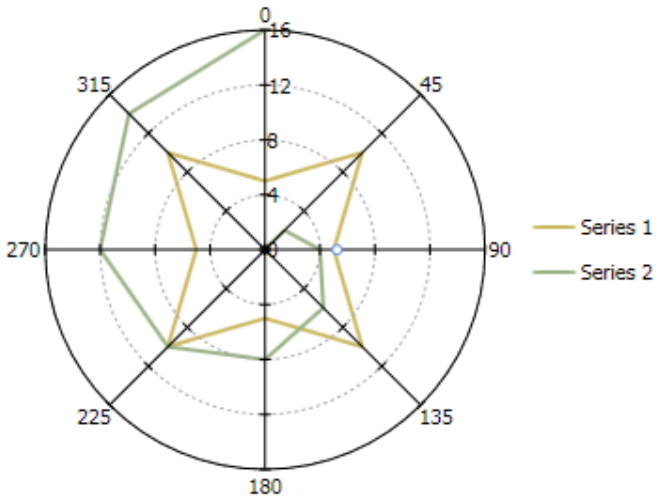
极坐标图表无法和其他图表类型组合在同一张图表区域。

下面的XAML标记为XYDataSeries指定数据值，并用此数据系列创建图像：

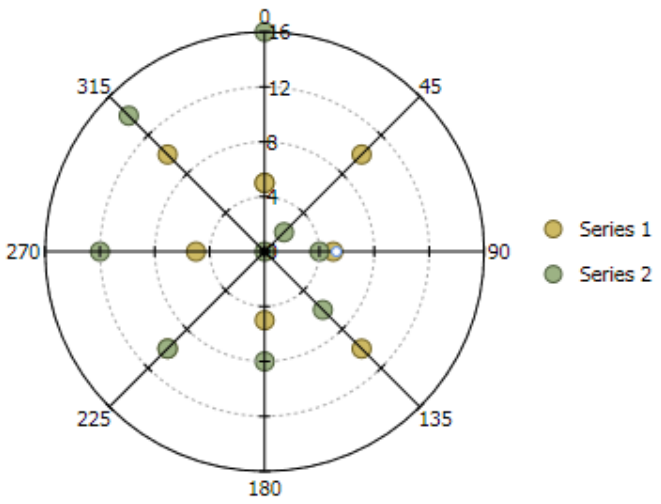
XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="PolarLinesSymbols">
  <c1chart:C1Chart.Data>
   <c1chart:ChartData>
    <c1chart:XYDataSeries Label="Series 1" Values="5 10 5 10 5 10 5 10 5"
                                XValues="0 45 90 135 180 225 270 315 0"/>
    <c1chart:XYDataSeries Label="Series 2" Values="0 2 4 6 8 10 12 14 16"
                               XValues="0 45 90 135 180 225 270 315 0"/>
   </c1chart:ChartData>
  </c1chart:C1Chart.Data>
  <c1chart:C1ChartLegend DockPanel.Dock="Right" /> </c1chart:C1Chart>
```

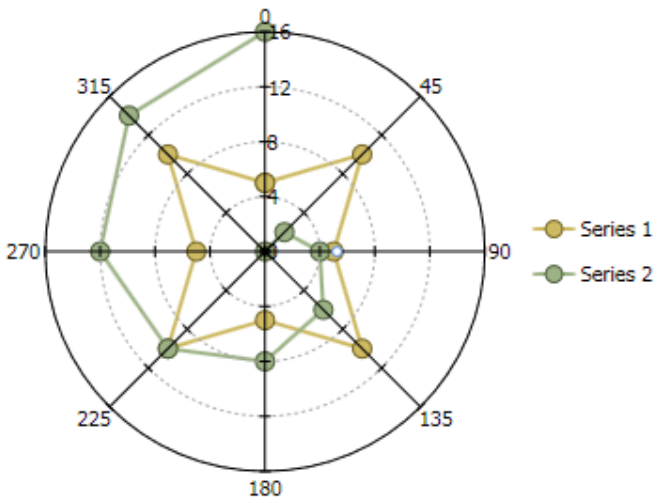
PolarLines



PolarSymbols



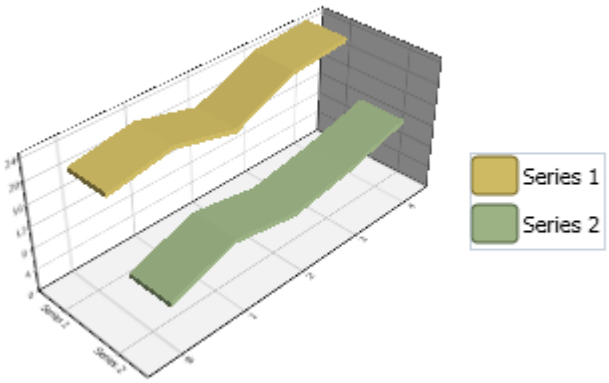
PolarLinesSymbols



3D Ribbon Chart

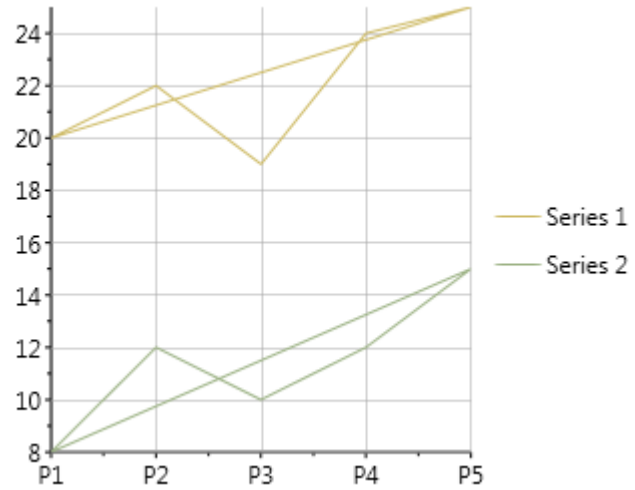
这个图表类型只在WPF中可用。

下面的图像展示了当您设置ChartType属性为Ribbon之后三维带状图的外观：

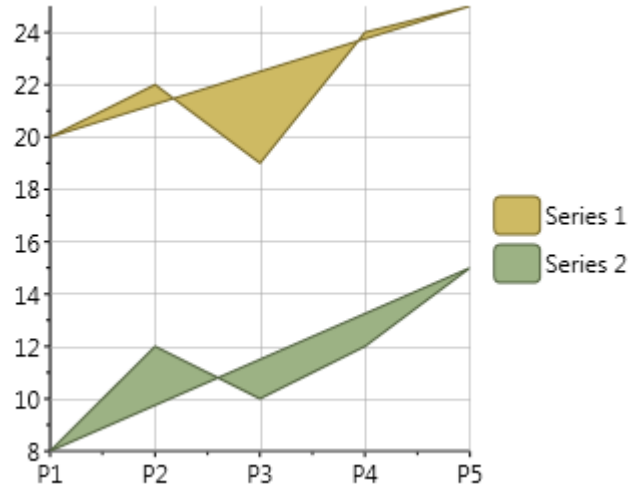


多边形图

下图展示了当您设置ChartType（在线文档 'ChartType属性'）属性的值为Polygon时，多边形图的外观：



下图展示了当您设置ChartType（在线文档 'ChartType属性'）属性的值为PolygonFilled时，多边形填充图的外观：



雷达图

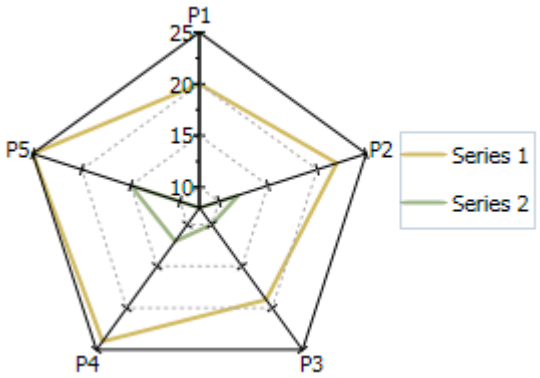
雷达图是一个极坐标图的变体。雷达图沿着雷达线方向绘制每一个数据集的Y值。如果数据具有N个独一无二的电，则图表平面将被分成N个夹角相等的段，且每隔360/N度的增量绘制一条雷达线（每条雷达线表示每一个数据点）。默认情况下，第一个点的雷达线垂直方向绘制（90度）。

雷达图的标签可以通过ChartData.ItemNames（在线文档 'ItemNames属性'）属性进行设置。这些标签位于每个径向线的末端。

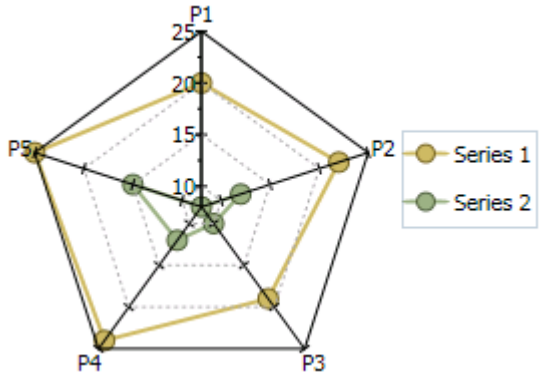
设置起始角度

PolarRadarOptions（在线文档 'PolarRadarOptions 类'）类的PolarRadarOptions.SetStartingAngle（在线文档 'SetStartingAngle 方法'）附加属性设置雷达图的起始角度。起始角让您顺时针旋转图表。例如，如果您设置SetStartingAngle属性的值为90，那么图表将沿着顺时针方向旋转90度。

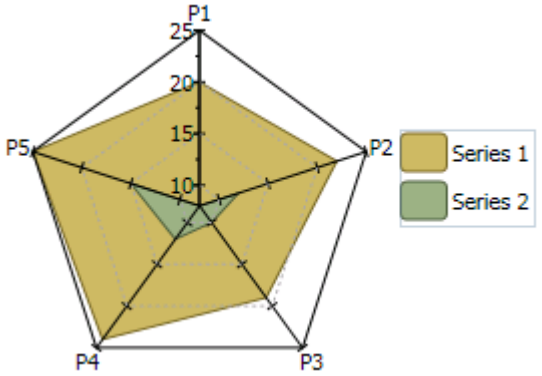
下图显示当您设置ChartType属性的值为Radar时雷达图的外观。



下图显示当您设置ChartType属性的值为RadarSymbols时带符号雷达图的外观。



下图显示当您设置ChartType属性的值为RadarFilled时填充雷达图的外观。



阶梯图

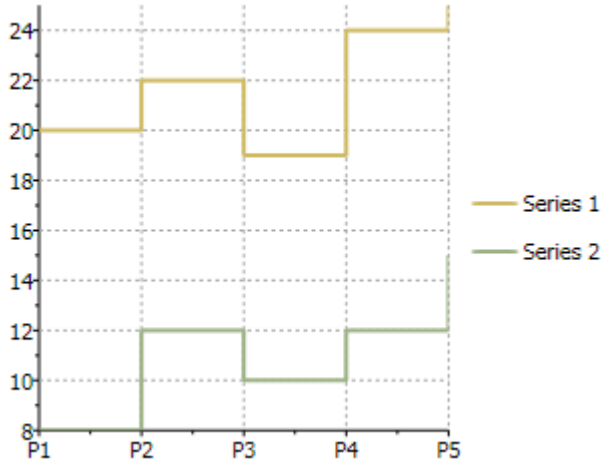
阶梯图是XY图的一种。阶梯图通常用于Y值发生离散的改变，且在某个特定的X值位置发生了一个突然的变化。

一个简单的，日常例子是按照时间记录的支票簿的曲线。当每一次存款发生，以及每一次书写支票支出时，支票登记的余额（Y值）都是立即发生变更，而不是按照时间的推移，缓缓的改变。在没有存入款项或书写支票的时候，余额（Y值）随着时间的流逝保持不变。
和折线图以及XY图类似，阶梯图的外观可以通过每一个系列的Connection以及Symbol属性进行自定义，包括修改颜色，符号尺寸以及线形的粗细。

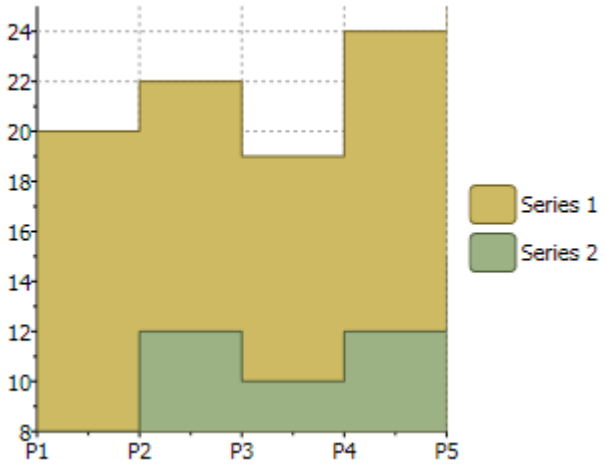
符号可以完全删除，以强调点之间的关系或包含到表示实际的数据值。如果存在数据空洞，阶梯图仍然可以按照预期表现，并且将已知的信息填充到系列线上数据空洞的X值。符号和线条将重复非空洞数据一次。

和大多数的XY样式的图表一样，阶梯图可以在适当的时候堆叠。

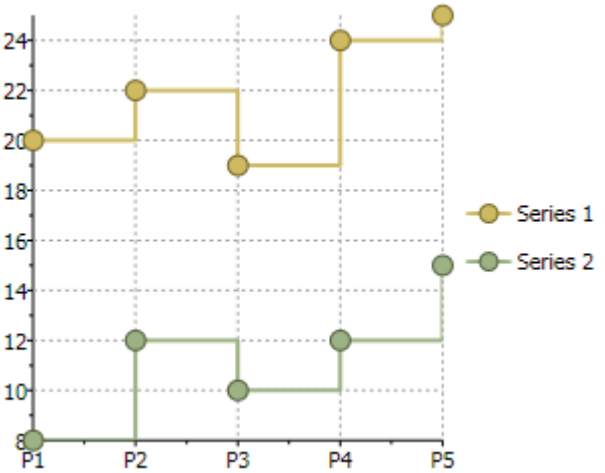
下图显示当您设置ChartType属性的值为Step时，阶梯图的外观：



下图显示当您设置ChartType属性的值为StepArea时，面积阶梯图的外观：



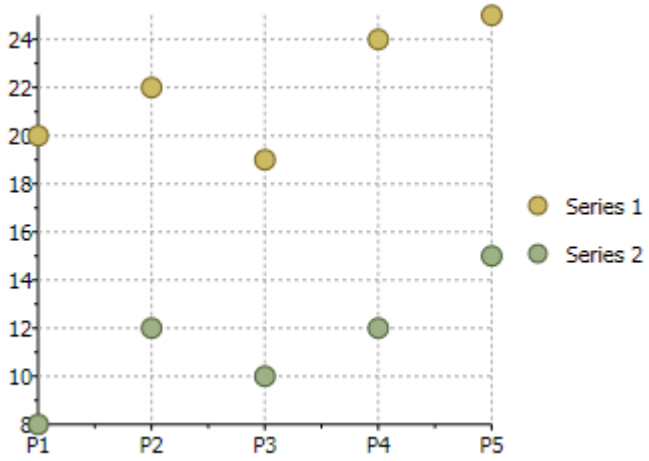
下图显示当您设置ChartType属性的值为StepSymbols时，符号阶梯图的外观：



散点图

XYPlot也称作散点图。

下图显示当您设置ChartType属性的值为XYPlot时，散点图的外观：



创建一个散点图

散点图，也称为XY图表，用于显示变量之间的关系。和目前为止我们介绍过的其他图表不同，在XY型图表中，每一个点具有两个数值。通过绘制一个相对于X轴一个相对于Y轴的值，图表显示一个变量对另一个的影响。

我们将继续利用早先我们创建的同样的数据继续我们的C1Chart之旅，但是这一次有所不同的是，我们将创建XY散点图，用来展示两个产品的收入之间的关系。例如，我们可能希望确定是否当Widget收入的提高和Gadgets收入的升高有关（也许这两种产品协作的比较好），或者是Widget收入的提高和Gadgets收入的降低有关（也许人们购买了其中一种产品就不需要另外一种）。

要做到这一点，我们将遵循相同的步骤。主要的区别是，这一次我们将添加XYDataSeries 对象至图表的Data.Children集合，而不是简单的DataSeries对象。用户获取数据的LINQ语句同样也更加的优雅和有趣。

步骤1）选择图表类型：

代码清除任何现有的系列，然后设置图表类型：

C#

```
public Window1()
{
   InitializeComponent();
   // 清除当前图表
   c1Chart.Reset(true);
   // 设置图表类型
   c1Chart.ChartType = ChartType.XYPlot;
```

步骤2）设置坐标轴：

因为我们现在创建XY系列，我们有两个数值轴（之前我们有一个标签轴和一个数值轴）。和我们之前做过的一样，我们将附加标题和格式至每一个坐标轴。我们还将和以前一样设置范围以及标注的格式。我们将使用AnnoAngle属性以沿着X轴旋转标注的标签，是的它们不会重叠在一起显示。

C#

```
// 获取坐标轴
  var yAxis = _c1Chart.View.AxisY;
  var xAxis = _c1Chart.View.AxisX;

  // 配置 Y 轴
  yAxis.Title = CreateTextBlock("Widget Revenues", 14, FontWeights.Bold);
  yAxis.AnnoFormat = "#,##0 ";
  yAxis.AutoMin = false;
  yAxis.Min = 0;
  yAxis.MajorUnit = 2000;
  yAxis.AnnoAngle = 0;

  // 配置 X 轴
  xAxis.Title = CreateTextBlock("Gadget Revenues", 14, FontWeights.Bold);
  xAxis.AnnoFormat = "#,##0 ";
  xAxis.AutoMin = false;
  xAxis.Min = 0;
  xAxis.MajorUnit = 2000;
  xAxis.AnnoAngle = -90; // rotate annotations
```

步骤3）添加一个或多个数据系列

再次，我们将使用早先定义的第二种data-privider方法。

C#

```
// 获取数据
  var data = GetSalesPerMonthData();
```

下一步，我们需要获得在每一天，Widgets以及Gadgets收入总额的相关XY值对。我们可以使用LINQ 直接从我们的数据中获取信息：

C#

```
 // 按照销售日期对数据进行分组
  var dataGrouped = from r in data
    group r by r.Date into g
    select new
    {
      Date = g.Key, // 按照日期分组
      Widgets = (from rp in g // 添加 Widget收入
        where rp.Product == "Widgets"
        select g.Sum(p => rp.Revenue)).Single(),
      Gadgets = (from rp in g // 添加 Gadget 收入
        where rp.Product == "Gadgets"
        select g.Sum(p => rp.Revenue)).Single(),
    };

    // 按照Widgets销售数据对数据进行排序
    var dataSorted = from r in dataGrouped
      orderby r.Gadgets
      select r;
```

第一个LINQ查询通过对数据进行日期分组查询数据的起始时间。然后，为每一个组，它将创建一个包含日期以及在该日期内每一种我们感兴趣的产品的收入总和。结果是包含三个属性的对象列表：Date, Widgets, 以及 Gadgets。这种类型的数据分组和聚和是LINQ所提供的强大功能。

第二个LINQ查询简单地按照Gadget收入数据按照日期进行排序。这些都是在X轴上绘制的值，我们希望它们以升序显示。如果我们仅显示符号（ChartType = XYPlot）那么绘制未排序的值看起来也不错，但是如果我们选择了其他的图表类型比如说折线图或者面积图，则看起来会相当凌乱。

一旦数据被正确地分组，总结，排序，我们所要做的事情就是创建一个单独的数据系列，并且将一组值的集合赋值给ValueSource属性而另一组值赋值给XValueSource属性：

C#

```
// 创建新的XYDataSeries
 var ds = new XYDataSeries();

 // 设置系列标签（显示在一个C1ChartLegend中）
 ds.Label = "Revenue:\r\nWidgets vs Gadgets";

 // 填充Y值
 ds.ValuesSource = (
   from r in dataSorted
   select r.Widgets).ToArray();

 // 填充X值
 ds.XValuesSource = (
   from r in dataSorted
   select r.Gadgets).ToArray();
 // 添加系列至图表
 c1Chart.ChartData.Children.Add(ds);
```

步骤4）调整图表的外观

再次，我们将通过设置主题属性来快速配置图表外观：

WPF

C#

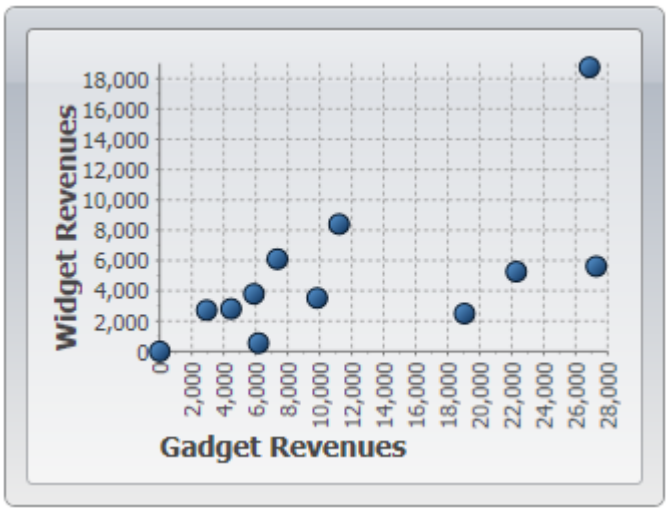
```
c1Chart.Theme = c1Chart.TryFindResource(new
ComponentResourceKey(typeof(C1.WPF.C1Chart.C1Chart), "Office2007Black")) as
ResourceDictionary;
        }
```

Silverlight

C#

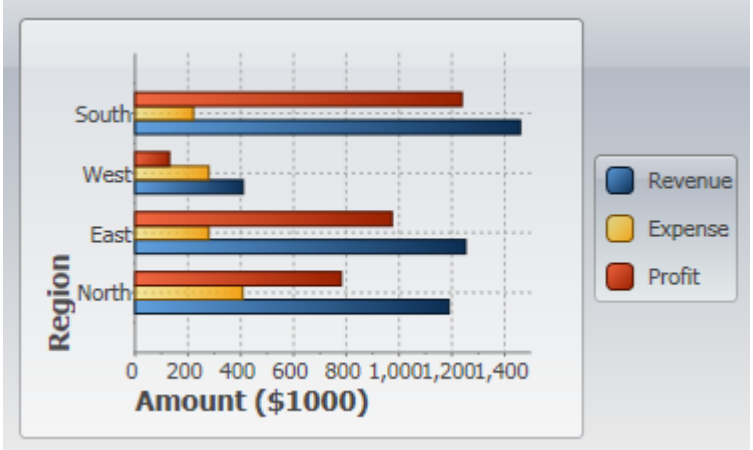
```
c1Chart.Theme = Theme.Office2007Black;
        }
```

完成的图表将显示Gadget 和Widget的收入之间的正相关关系。



简单图表

简单图表指的是每一个数据点具有一个单一的与之关联的数值的图表。一个典型的例子是一个图表显示不同地区的销售数据，类似于下图：



在我们可以创建任何图表之前，我们需要生成将数据显示为图表的数据。这里是一些代码来创建我们需要的数据。

> **注意：**在这个代码中没有什么和图表相关的逻辑，这只是一些一般的数据。我们将利用这些数据建立时间系列和XY图表以及在接下来的话题中继续使用。

C#

```
// 简单用来保存虚拟销售数据的类型
public class SalesRecord
{
  // 属性
  public string Region  { get; set; }
  public string Product { get; set; }
  public DateTime Date  { get; set; }
  public double Revenue { get; set; }
```

```
  public double Expense { get; set; }
  public double Profit  { get { return Revenue - Expense; } }
  // 构造器一
  public SalesRecord(string region, double revenue, double expense)
  {
    Region = region;
    Revenue = revenue;
    Expense = expense;
  }
  // 构造器二
  public SalesRecord(DateTime month, string product,    double revenue, double
expense)
  {
    Date = month;
    Product = product;
    Revenue = revenue;
    Expense = expense;
  }
}
// 返回一个列表，每一个区域一个SalesRecord记录
List<SalesRecord> GetSalesPerRegionData()
{
  var data = new List<SalesRecord>();
  Random rnd = new Random(0);
  foreach (string region in "North,East,West,South".Split(','))
  {
    data.Add(new SalesRecord(region, 100 + rnd.Next(1500), rnd.Next(500)));
  }
  return data;
}
// 返回一个列表，每一个产品一条SalesRecord记录
// 在过去的12个月期间内

List<SalesRecord> GetSalesPerMonthData()

{
  var data = new List<SalesRecord>();
  Random rnd = new Random(0);
  string[] products = new string[] {"Widgets", "Gadgets", "Sprockets" };
  for (int i = 0; i < 12; i++)
  {
    foreach (string product in products)
    {
      data.Add(new SalesRecord(
        DateTime.Today.AddMonths(i - 24),
        product,
        rnd.NextDouble() * 1000 * i,
        rnd.NextDouble() * 1000 * i));
      }
    }
    return data;
```

```
    }
}
```

请注意，SalesData 类为Public类型。这是数据绑定所必需的。

在创建图表时，我们将遵循以下四个主要步骤：

步骤1）选择图表类型：

下面的代码将清除任何现有的序列，然后设置图表类型：

C#

```
public Window1()
{
    InitializeComponent();
    // 清除当前图表
     c1Chart.Reset(true);
    // 设置图表类型
   c1Chart.ChartType = ChartType.Bar;
}
```

步骤2）设置坐标轴：

我们将开始获得对这两个坐标轴的引用。在大多数图表中，水平轴（X轴）显示关联到每一个点的标签，而垂直轴（Y轴）则显示数值。条形图表类型是一个例外，它显示水平方向的长条形。对于这个图表类型，标签显示在Y轴上而数值则显示在X轴上：

下一步我们将标题指定给坐标轴。坐标轴标题是UIElement对象而不是简单的文本。这意味着你对标题的格式有完全的控制灵活性。事实上，您可以在坐标轴标题上使用带有按钮，表格或者图像的复杂元素。在本示例中，我们将使用一个简单的TextBlock元素，该元素由一个叫做CreateTextBlock的方法创建，我们将在稍后介绍这个方法。

我们还将配置值坐标轴从零开始，并在刻度标记旁边的标注上使用带有千位分隔符的格式显示标注：

C#

```
// 配置标签坐标轴
  labelAxis.Title = CreateTextBlock("Region", 14, FontWeights.Bold);

  // 配置数值坐标轴
_c1Chart.View.AxisX.Title = CreateTextBlock("Amount ($1000)", 14, FontWeights.Bold);
  c1Chart.View.AxisX.AutoMin = false;
  c1Chart.View.AxisX.Min = 0;
  c1Chart.View.AxisX.MajorUnit = 200;
  c1Chart.View.AxisX.AnnoFormat = "#,##0 ";
```

步骤3）添加一个或多个数据系列

我们开始这一步，通过之前列出的方法获取数据：

C#

```
// 获取数据
  var data = GetSalesPerRegionData();
```

接下来，我们要显示沿标签坐标轴的区域。为此，我们将使用一个LINQ语句检索每个记录的Region属性。然后结果被转换为数组赋给ItemNames属性。

C#

```
// 沿着标签坐标轴显示区域信息
  c1Chart.ChartData.ItemNames = (from r in data select r.Region).ToArray();
```

请注意，这里是如何使用LINQ使得代码变得简洁明了的。因为我们的样本数据只包含一个记录区域，所以事情变得更简单了。在一个更接近现实的情况下，会有每个地区的几个记录，我们会用一个更复杂的LINQ语句对每个区域的数据进行分组。

现在我们可以创建将被添加到图表的实际DataSeries对象。我们将创建三个系列："Revenue"，"Expenses"，以及"Profit":

C#

```
// 添加 Revenue 系列
  var ds = new DataSeries();
  ds.Label = "Revenue";
  ds.ValuesSource = (from r in data select r.Revenue).ToArray();
c1Chart.Data.Children.Add(ds);
  // 添加 Expense 系列
  ds = new DataSeries();
  ds.Label = "Expense";
  ds.ValuesSource = (from r in data select r.Expense).ToArray();
  c1Chart.ChartData.Children.Add(ds);
  // 添加 Profit 系列
  ds = new DataSeries();
  ds.Label = "Profit";
  ds.ValuesSource = (from r in data select r.Profit).ToArray();
  c1Chart.Data.Children.Add(ds);
```

对于每个系列，代码创建了一个新的DataSeries对象，并设置其Label属性。标签是可选的；如果提供，它将在任何与此图表相关的C1ChartLegend对象上进行显示。接下来，一个LINQ语句用于从数据源中检索值。结果将设置给数据系列对象的ValuesSource属性。最后，将数据序列添加到图表的Children 集合中。

再次，注意LINQ的使用使代码变得如此地简洁、自然。

步骤4）调整图表的外观

我们将使用Theme属性快速配置图表外观：

WPF

C#

```
c1Chart.Theme = _c1Chart.TryFindResource(new
ComponentResourceKey(typeof(C1.WPF.C1Chart.C1Chart),"Office2007Black")) as
ResourceDictionary;}
```

Silverlight

C#

```
// 设置主题
c1Chart.Theme = _c1Chart.TryFindResource(new
ComponentResourceKey(typeof(C1.Silverlight.C1Chart.C1Chart),"Office2007Black")) as
ResourceDictionary;}
```

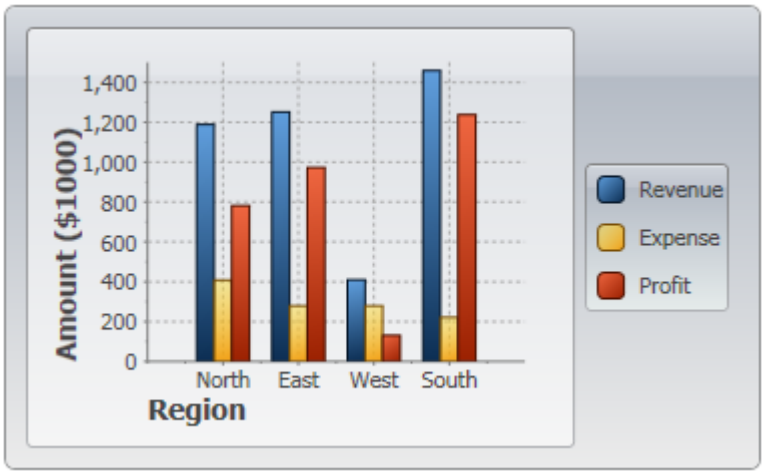
是否还记得我们在设置坐标轴时使用到了一个CreateTextBlock辅助方法。这里是此方法的具体定义：

C#

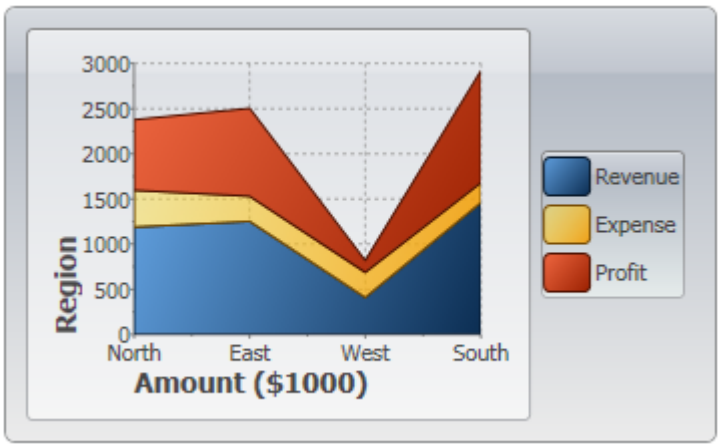
```
TextBlock CreateTextBlock(string text, double fontSize, FontWeight fontWeight)
{
  var tb = new TextBlock();
  tb.Text = text;
  tb.FontSize = fontSize;
  tb.FontWeight = fontWeight;
  return tb;
}
```

以上代码将生成简单的数值图表。你可以通过修改ChartType属性为任意余下的简单的图表类型值的以尝试效果：Bar，AreaStacked，Pie以创建不同类型的图表。注意，如果你改变ChartType为Column，您将需要在Y轴上显示数据的标签，所以你会用到AxisY。其结果应该是类似于下面的图像：

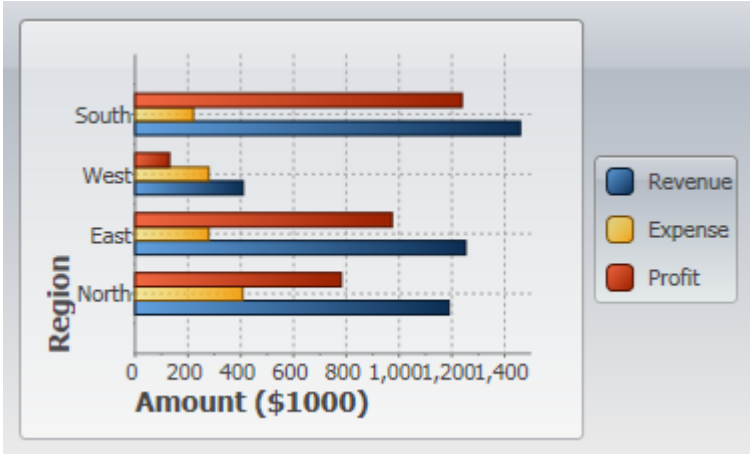
ChartType.Column



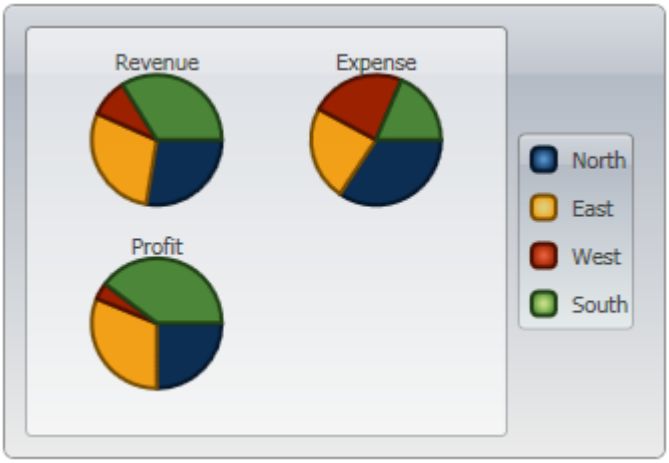
ChartType.AreaStacked



ChartType.Bar



ChartType.Pie



特殊图表类型和组合图

使用XAML标记或代码，您可以创建组合图表。这将给您的C1Chart（在线文档 'C1Chart 类'）应用程序更大的灵活性。

添加一个条形系列以及一个折线系列

为了以编程方式添加条形系列以及一个折线系列，可以使用下面的代码：

C#

```
chart.Data.Children.Add(new XYDataSeries() {
       ChartType=ChartType.Column,
       XValuesSource = new double[] {1,2,3 },
       ValuesSource = new double[] {1,2,3 } });

     chart.Data.Children.Add(new XYDataSeries() {
```

```
        ChartType = ChartType.Line,
        XValuesSource = new double[] { 1, 2, 3 },
        ValuesSource = new double[] { 3, 2, 1 } });
```

柱形折线图

使用不同的数据系列模板，很容易生成各种图表类型组合。

该图表可以通过设置DataSeries.ChartType创建。

XAML

```
<c1chart:C1Chart Name="c1chart1">
   <c1chart:C1Chart.Data>
     <c1chart:ChartData >
       <!-- 第一个系列的默认外观（柱形） -->
       <c1chart:DataSeries Label="series 1" Values="0.5 2 3 4" />
       <!-- 第二个系列中的星星符号通过折线连接 -->
       <c1chart:DataSeries Label="series 2" Values="1 3 2 1"
        ChartType="LineSymbols" SymbolMarker="Star4" />
     </c1chart:ChartData>
   </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

创建一个高斯曲线

高斯或正态分布曲线是用来显示一个随机变量的值的概率分布。

使用C1Chart创建高斯曲线，使用下面的代码：

C#

```
  // 创建和添加代表高斯函数的图表数据系列
    //    y(x) = a * exp( -(x-b)*(x-b) / (2*c*c))
    // 位于从 x1 到 x2 的区间

    void CreateGaussian(double x1, double x2, double a, double b, double c)
    {
      // 点的个数
      int cnt = 200;
      var xvals = new double[cnt];
      var yvals = new double[cnt];

      double dx = (x2 - x1) / (cnt-1);

      for (int i = 0; i < cnt; i++)
      {
        var x = x1 + dx * i;
        xvals[i] = x;
        x =  (x - b) / c;
        yvals[i] = a * Math.Exp(-0.5*x*x);
      }
```

```
      var ds = new XYDataSeries()
      {
        XValuesSource = xvals,
        ValuesSource = yvals,
        ChartType = ChartType.Line
      };

      chart.Data.Children.Add(ds);
    }
```

创建一个帕累托图表

帕累托图表突出显示了一组数据中最重要的因素。

为创建一个帕累托图，请使用下面的XAML标记：

XAML

```
<c1chart:C1Chart Name="c1Chart1">
   <c1chart:C1Chart.View>
     <c1chart:ChartView>
       <c1chart:ChartView.AxisX>
         <c1chart:Axis AnnoAngle="-75" MajorGridStroke="Gray"/>
       </c1chart:ChartView.AxisX>
       <!-- 标准（默认）左侧Y轴 -->
       <c1chart:ChartView.AxisY>
         <c1chart:Axis Min="0" Max="50" Title="Frequency"
MajorGridStroke="Gray"/>
       </c1chart:ChartView.AxisY>
        <!-- 辅助（右侧）Y轴 -->
       <c1chart:Axis Name="ay2" AxisType="Y" Position="Far" AnnoFormat="p"
           Min="0" Max="1" />
     </c1chart:ChartView>
   </c1chart:C1Chart.View>
   <c1chart:C1Chart.Data>
     <c1chart:ChartData>
       <c1chart:ChartData.ItemNames>Documents Quality Packaging Delivery
Other</c1chart:ChartData.ItemNames>
       <c1chart:DataSeries Values="40 30 20 5 5" />
       <c1chart:DataSeries AxisY="ay2" Values="0.4 0.7 0.9 0.95 1.0"
ChartType="LineSymbols" />
     </c1chart:ChartData>
   </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

图表功能

以下几个部分介绍了图表功能。

动画

几乎所有的绘图区元素都可以使用内置的动画。使用内置动画选项简化了为C1Chart控件的绘图区元素创建各种不同的视觉动画效果的设置。包含在PlotElementAnimation类中的属性如下所示：

属性	描述
IndexDelay	一个附加属性，允许您指定按照元素点的索引不同，动画的延迟。
Storyboard	获取或设置应用到绘图区元素的storyboard。
SymbolStyle	获取或设置在 storyboard 开始之前应用到绘图区元素的符号样式。

对C1Chart控件应用动画还涉及到ChartData.LoadAnimation。

使用内置的动画选项设置动画效果，您可以使用下面的XAML标记：

XAML

```
<c1chart:C1Chart Name="chart">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:ChartData.LoadAnimation>
                        <!-- 加载动画 -->
                        <c1chart:PlotElementAnimation>
                            <!-- 初始样式：不可见  -->
                            <c1chart:PlotElementAnimation.SymbolStyle>
                                <Style TargetType="c1chart:PlotElement">
                                    <Setter Property="Opacity" Value="0" />
                                </Style>
                            </c1chart:PlotElementAnimation.SymbolStyle>
                            <c1chart:PlotElementAnimation.Storyboard>
                                <Storyboard >
                                   <!-- 按照索引值不同延迟显示元素 -->
                                    <DoubleAnimation
                  Storyboard.TargetProperty="Opacity"
                  c1chart:PlotElementAnimation.IndexDelay="0.5"
                  To="1" Duration="0:0:1" />
                                </Storyboard>
                            </c1chart:PlotElementAnimation.Storyboard>
                        </c1chart:PlotElementAnimation>
                     </c1chart:ChartData.LoadAnimation>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
        </c1chart:C1Chart>
```

然后你可以将下面的代码直接插入在InitializeComponent()方法中：

C#

```
var rnd = new Random();
            chart.MouseLeftButtonDown += (s, e) =>
            {
                chart.Data.Children.Clear();
                // 创建新的数据
                var vals = new double[rnd.Next(5, 10)];
                for (int i = 0; i < vals.Length; i++)
                    vals[i] = rnd.Next(0, 100);
                chart.Data.Children.Add(new DataSeries() { ValuesSource = vals });
            };
```

当运行您的应用程序时，数据将在鼠标单击时加载或重新加载，显示在XAML标记中设置的动画效果。

创建自定义动画

几乎所有的绘图区元素可以使用标准WPF动画做为动画效果。下面的样式是将“奔跑的蚂蚁”动画添加到鼠标指针下的元素的修改样式。

XAML

```
WPF版Chart/ Chart功能 /动画
创建自定义动画
几乎所有的绘图区元素可以使用标准WPF动画做为动画效果。下面的样式是将“奔跑的蚂蚁”动画添加到鼠标指针下的元素的修改样式。


<Style x:Key="mouseOver" TargetType="{x:Type c1c:PlotElement}">
   <!-- 默认的黑色边框 -->
   <Setter Property="Stroke" Value="Black" />
   <Style.Triggers>
   <!-- 当鼠标悬停经过元素时，显示粗的红色边框  -->
          <Trigger Property="IsMouseOver" Value="true">
                 <Setter Property="Stroke" Value="Red" />
                 <Setter Property="StrokeThickness" Value="2" />
                 <Setter Property="StrokeDashArray" Value="2,2" />
                 <Setter Property="Canvas.ZIndex" Value="1" />
                 <Trigger.EnterActions>
                        <!-- 启动动画 -->
                        <BeginStoryboard >
                               <Storyboard>
                               <DoubleAnimation
Storyboard.TargetProperty="StrokeDashOffset"
                               From="0" To="8" RepeatBehavior="Forever"
Duration="0:0:0.5"/>
                               </Storyboard>
                        </BeginStoryboard>
                 </Trigger.EnterActions>
          </Trigger>
   </Style.Triggers>
</Style>
```

图表中的每一个系列由PlotElement对象组合而成，每一个对象表示位于系列中的单独的符号，连接线，区域，饼图切块，等等。具体的PlotElement类型取决于图表类型。

您可以通过附加Storyboard对象至绘图区元素的方式向图表添加动画。这通常在DataSeries.PlotElementLoaded事件中完成，该事件在创建并添加至数据系列之后触发。

坐标轴

标注

沿每一个坐标轴的标注是任何图表的重要组成部分。图表通过基于输入在BubbleSeries，DataSeries，HighLowOpenCloseSeries，HighLowSeries，或者XYDataSeries对象的数据/值的数值，对坐标轴进行标注。坐标轴的标注始终显示基本文本，而不应用任何的格式。

图表会自动产生最自然的注释，即使是图表数据发生了变化。

坐标轴标注格式

您可以通过AnnoFormat（在线文档 'AnnoFormat 属性'）属性控制X或者Y轴上显示的值的标注格式。

设置AnnoFormat（在线文档 'AnnoFormat属性'）属性为某个.NET Framework 组合格式字符串将格式化输入到该属性的值。更多关于您可以在AnnoFormat属性中使用的标准数值格式字符串的更多信息，请参见标准数值格式字符串。

DateTime格式字符串

DateTime格式字符串被分为两类：

- 标准日期时间格式字符串
- 自定义日期时间格式字符串

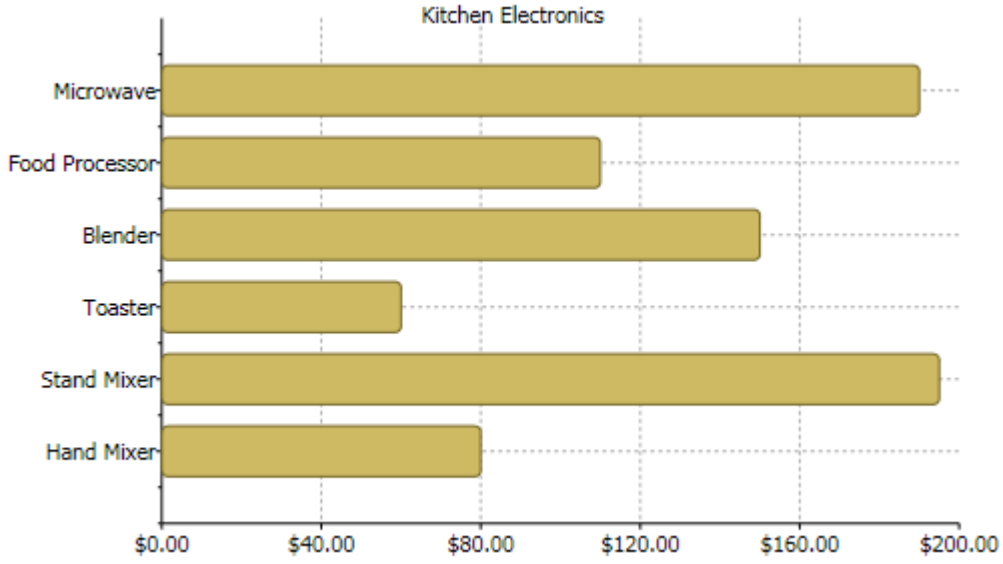
数字格式字符串

- 标准数字格式字符串
- 自定义数字格式字符串

自定义数字格式字符串

您还可以使用自定义数字格式字符串来自定义格式字符串。

为使用AnnoFormat属性，请为其指定一个标准或自定义格式字符串。例如，下面的图表的AnnoFormat属性设置为“c”，用来改变整个数值显示为金融货币格式。



XAML

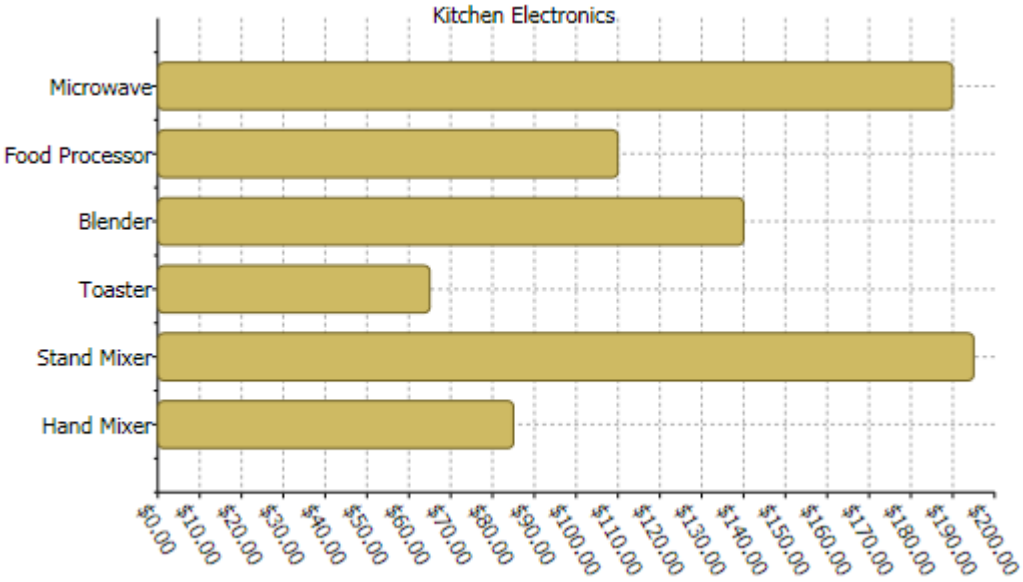
```
<c1chart:C1Chart.View>
                <c1chart:ChartView>
                    <c1chart:ChartView.AxisX>
                        <c1chart:Axis Min="0" AnnoFormat="c" AutoMin="false"
AutoMax="false" Max="200" />
                    </c1chart:ChartView.AxisX>
                </c1chart:ChartView>
            </c1chart:C1Chart.View>
```

C#

```
// 金融格式
   c1Chart1.View.AxisX.AnnoFormat = "c";
   c1Chart1.View.AxisX.Min = 0;
```

旋转坐标轴标签

通过AnnoAngle（在线文档 'AnnoAngle 属性'）属性按照指定的角度逆时针转动坐标轴标注。如果X-轴挤满了各种标注，则您会发现这个属性尤其有用。将标注沿着正负30或60度的角度旋转可以在水平坐标轴的一个狭窄的空间中放置下更多数量的标注。利用AnnoAngle属性，X-轴标注将不再发生重叠，如下图所示：



XAML

```
<c1chart:C1Chart.View>
                <c1chart:ChartView>
                    <c1chart:ChartView.AxisX>
                        <c1chart:Axis Min="0" MajorUnit="10" AnnoFormat="c"
AutoMin="false" AutoMax="false" Max="200" AnnoAngle="60"  />
                    </c1chart:ChartView.AxisX>
                </c1chart:ChartView>
            </c1chart:C1Chart.View>
```

C#

```
// 金融格式
c1Chart1.View.AxisX.AnnoFormat = "c";
c1Chart1.View.AxisX.Min = 0;
c1Chart1.View.AxisX.AnnoAngle = "60";
```

自定义坐标轴标注

在某些情况下，您可能需要创建自定义坐标轴标注。下面的场景中，创建自定义坐标轴标注会比较有用：

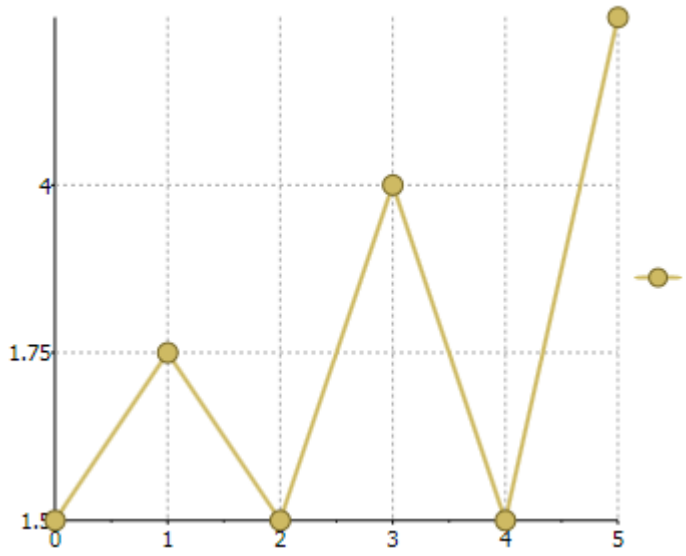
- 当ItemsSource（在线文档 'ItemsSource属性'）属性是一个数值或日期类型的值的集合，且图表使用这些值做为坐标轴标签。下面的代码使用ItemsSource属性创建 Y-轴的自定义标签：

  C#

  ```
  c1Chart1.Reset(true);
          c1Chart1.Data.Children.Add(
            new DataSeries() { ValuesSource = new double[] { 1, 2, 1, 3, 1, 4 }
  ```

```
});
        c1Chart1.ChartType = ChartType.LineSymbols;
        c1Chart1.View.AxisY.ItemsSource = new double[] { 1.25, 1.5, 1.75, 4 };
```

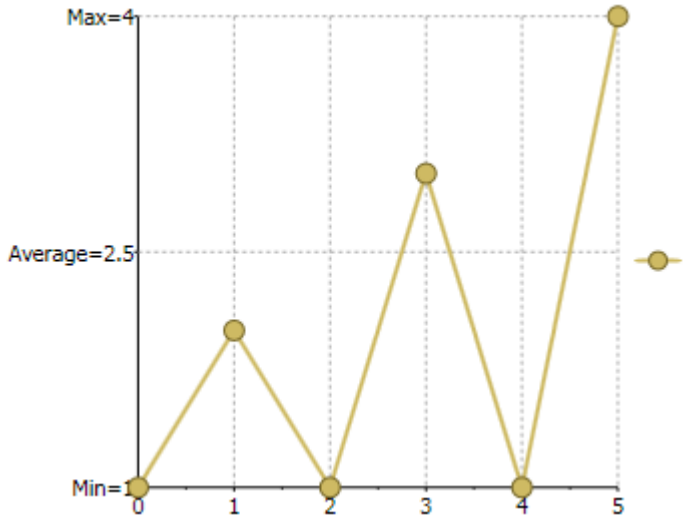
下面是添加前面的代码后，该图表的外观：



C#

```
c1Chart1.Reset(true);

        c1Chart1.Data.Children.Add(
          new DataSeries() { ValuesSource = new double[] { 1, 2, 1, 3, 1, 4 }
});
        c1Chart1.ChartType = ChartType.LineSymbols;

        c1Chart1.View.AxisY.ItemsSource = new List<KeyValuePair<object,double>>
          { new KeyValuePair<object,double>("Min=1", 1),
            new KeyValuePair<object,double>("Average=2.5", 2.5),
            new KeyValuePair<object,double>("Max=4", 4)};
```

下面是添加前面的代码后，该图表的外观：

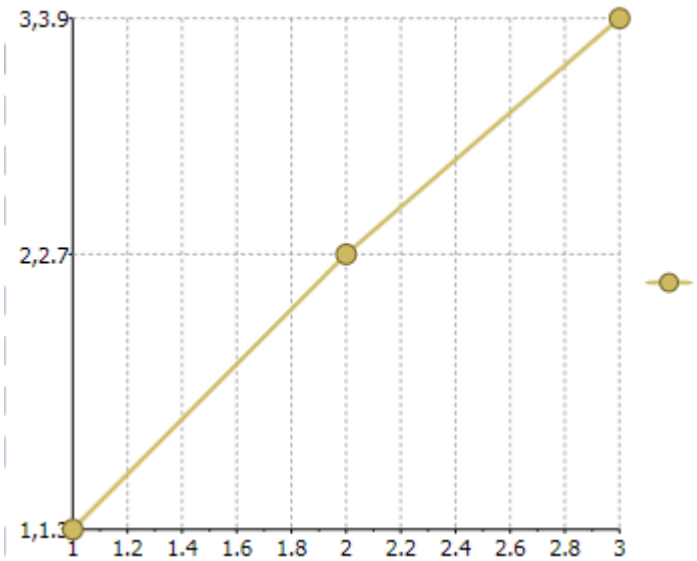


- 您可以使用ItemsValueBinding（在线文档 'ItemsValueBinding 属性'）属性以及ItemsLabelBinding属性以创建坐标轴标签，使用任意集合作为数据源，如下面的代码：

C#

```
c1Chart1.Reset(true);

        Point[] pts = new Point[] { new Point(1, 1.3), new Point(2, 2.7), new
Point(3, 3.9) };
        c1Chart1.DataContext = pts;
        c1Chart1.ChartType = ChartType.LineSymbols;

        c1Chart1.View.AxisY.ItemsSource = pts;
        c1Chart1.View.AxisY.ItemsValueBinding = new Binding("Y");
        c1Chart1.View.AxisY.ItemsLabelBinding = new Binding();
```

下面是添加前面的代码后，该图表的外观：



创建一个标注模板

为了通过AnnoTemplate（在线文档'AnnoTemplate属性'）属性创建一个自定义标注，请使用以下XAML标注或者C#代码：

XAML

```
<c1chart:ChartView.AxisX>
      <c1chart:Axis>
        <c1chart:Axis.Resources >
          <local:ColorConverter x:Key="clrcnv" />
        </c1chart:Axis.Resources>
        <c1chart:Axis.AnnoTemplate>
          <DataTemplate>
            <TextBlock Width="25" TextAlignment="Center"
             Text="{Binding Path=Value}"
             Foreground="{Binding Converter={StaticResource clrcnv}}"/>
          </DataTemplate>
        </c1chart:Axis.AnnoTemplate>
      </c1chart:Axis>
    </c1chart:ChartView.AxisX>
```

C#

```
public class ColorConverter : IValueConverter {
   int cnt = 0;
   public object Convert(object value, Type targetType,
     object parameter, CultureInfo culture)
   {
     //DataPoint dpt = (DataPoint)value;
     //交替画刷颜色

     return cnt++ % 2 == 0 ? Brushes.Blue : Brushes.Red;
   }

   public object ConvertBack(object value, Type targetType,
    object parameter, CultureInfo culture)
   {
     return null;
   }
}
```

在Chart相反的一侧显示坐标轴以及刻度线

为了在图表相反的一侧显示水平坐标轴以及标注，您可以使用辅助坐标轴并将坐标轴放置在顶部和标题一起的位置，如下面的代码所示：

C#

```
c1Chart1.View.Axes.Add(new Axis()
      {
        AxisType = AxisType.X,
        Position = AxisPosition.Far,
        ItemsSource = new string[] { ""},
        Title = "Axis title",
      });
```

坐标轴线

坐标轴线对于Y轴而言是从起始值到结束值的水平方向的线，对于X轴而言是从起始值到结束值的垂直方向的线。Z轴线用于三维图表。

您可以使用Axis.Foreground或者ShapeStyle.Stroke属性以应用颜色至坐标轴线。请注意，Axis.Foreground属性将覆盖ShapeStyle.Stroke属性的值。

您可以通过StrokeStartLineCap 以及 StrokeEndLineCap 属性指定坐标轴线形的起始点和结束点的形状。

副坐标轴

IsDependent （在线文档 'IsDependent 属性'）属性允许链接辅助坐标轴到某一个主坐标轴（AxisX或AxisY，取决于AxisType）。副坐标轴轴始终具有和主坐标轴相同的最小值和最大值。

新的属性 DependentAxisConverter （在线文档 'DependentAxisConverter属性'）以及委托Axis.AxisConverter指定用来从主坐标轴到副坐标轴坐标系的转换。

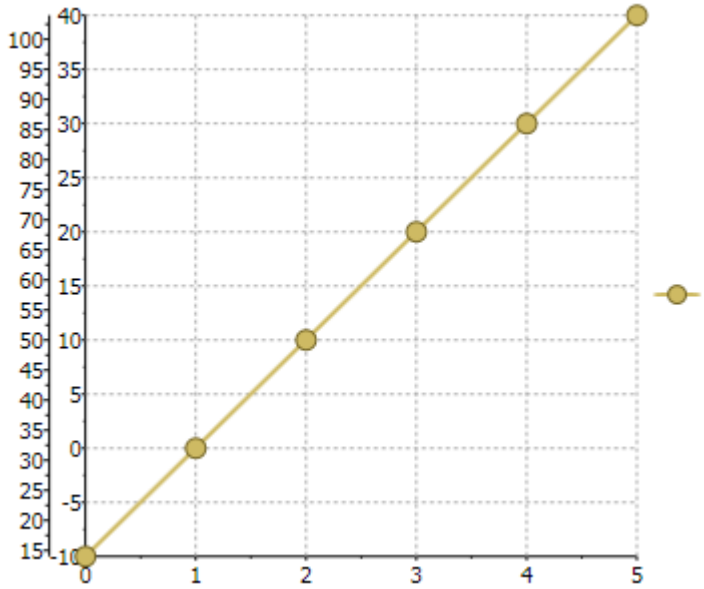
下面的代码创建一个副Y轴：

C#

```
c1Chart1.Reset(true);
        c1Chart1.Data.Children.Add(
          new DataSeries() { ValuesSource = new double[] { -10, 0, 10, 20, 30, 40 }
});
        c1Chart1.ChartType = ChartType.LineSymbols;
        Axis axis = new Axis() { AxisType = AxisType.Y, IsDependent =true};
        //摄氏度-> 华氏度
        axis.DependentAxisConverter = (val) => val * 9 / 5 + 32;
```

```
        c1Chart1.View.Axes.Add(axis);
```

下图展示了显示华氏度的从属（最左侧）Y-轴关联到显示摄氏度的主坐标轴Y轴：



坐标轴位置

您可以通过设置Position（在线文档 'Position 属性'）属性为Near或Far的值指定坐标轴的位置。对于垂直方向的坐标轴，Axis.Position.Near 对应左侧位置，而
Axis.Position.Far则对应右侧位置。对于水平方向的坐标轴，Axis.Position.Near 对应底部位置，为Axis.Position.Far 对应顶部位置。

坐标轴范围

通常情况下，图形显示它所包含的所有数据。然而，可以显示通过固定的坐标轴范围，使得仅显示部分图表的数据。

图表通过分析最小值和最大值以及增量数值确定每一个坐标轴的范围。设置Min（在线文档'Min 属性'）以及Max（在线文档'Max属性'），AutoMin（在线文档'AutoMin属性'），以及AutoMax（在线文档'AutoMax 属性'）属性允许自定义这一过程。

坐标轴最小和最大值

使用Min以及Max属性可以在特定的坐标轴上框定图表值的范围。如果图表的X轴的值从0到100，然后设置最小为0，最大值为10将只显示值最大不超过10的值。

图表还可以自动计算最小值和最大值。如果AutoMax以及AutoMin属性设置为True，那么图表将自动格式化坐标轴的值以适应当前的数据集。

坐标轴原点

您可以通过Origin（在线文档'Origin 属性'）属性指定坐标轴远点，如下所示：

```
C#
```

```
{
          c1Chart1.Reset(true);
          c1Chart1.Data.Children.Add(new XYDataSeries()
                    {
                      ValuesSource = new double[] { -1, 2, 0, 2, -2 },
                      XValuesSource = new double[] { -2, -1, 0, 1, 2 }
                    };
          c1Chart1.View.AxisX.Origin = 0;
          c1Chart1.View.AxisY.Origin = 0;
          c1Chart1.ChartType = ChartType.LineSymbols;
        }));
```

坐标轴标题

添加一个标题来声明沿着该坐标轴图表展示的内容。例如，如果你的数据包括测量数据，那么在坐标轴上包含该测量的单位（克，米，升等）将有助于最终用户理解图表的内容。坐标轴标题可以添加到面积图、散点图、条形图、HiLoOpenClose图或者蜡烛图。

坐标轴标题是UIElement对象而不是简单的文本。这意味着你对标题的格式的控制有完全的灵活性。事实上，你可以在坐标轴标题上使用包括按钮、表格或图像的复杂元素。

以编程方式设置坐标轴标题

C#

```
//设置坐标轴标题
   c1Chart1.View.AxisY.Title= new TextBlock() { Text = "Kitchen Electronics" };
   c1Chart1.View.AxisX.Title = new TextBlock() { Text = "Price" };
```

使用XAML代码设置坐标轴标题

XAML

```
<c1chart:C1Chart >
      <c1chart:C1Chart.View>
        <c1chart:ChartView>
          <c1chart:ChartView.AxisX>
            <c1chart:Axis>
              <c1chart:Axis.Title>
                <TextBlock Text="Price" TextAlignment="Center" Foreground="Crimson"/>
              </c1chart:Axis.Title>
            </c1chart:Axis>
          </c1chart:ChartView.AxisX>
          <c1chart:ChartView.AxisY>
            <c1chart:Axis>
              <c1chart:Axis.Title>
                <TextBlock Text="Kitchen Electronics" TextAlignment="Center"
Foreground="Crimson"/>
              </c1chart:Axis.Title>
```

```
            </c1chart:Axis>
          </c1chart:ChartView.AxisY>
        </c1chart:ChartView>
      </c1chart:C1Chart.View>
    </c1chart:C1Chart>
```

刻度线

图表自动设置了主要的和次要的刻度线。自定义刻度间隔或属就像操作一组属性那样容易。

MajorUnit以及MinorUnit属性设置坐标轴刻度线的状态。为了消除图表中可能存在的杂波，您可以通过指定显示分类标签的间隔，在分类（X）轴上显示较少数量的标签或者刻度线，或者还可以指定显示在两个刻度线之间的分类数。

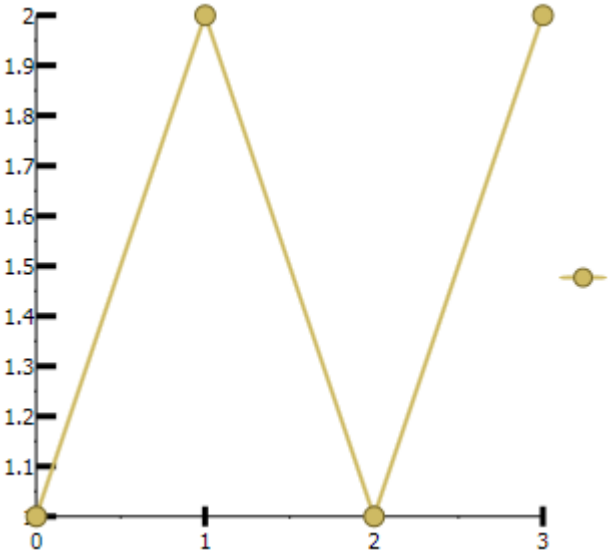
主刻度线重叠

您可以通过为MajorTickOverlap（在线文档 'MajorTickOverlap 属性'）属性指定一个从 0 到 1 的值，决定主刻度线标记的重叠程度。
默认值是0，这意味着没有重叠。当重叠是1，整个刻度线位于绘图区域内。对于 X-轴，当年您增加MajorTickOverlap属性的值时，刻度线向上移动，而减少该属性的值时，刻度线向下移动。对于 Y-轴，当您增加MajorTickOverlap属性的值时，刻度线向左移动。

C#

```
c1Chart1.Reset(true);
c1Chart1.Data.Children.Add(
new DataSeries() { ValuesSource = new double[] { 1, 2, 1, 2 } });
c1Chart1.ChartType = ChartType.LineSymbols;
c1Chart1.View.AxisX.MajorGridStrokeThickness = 0;
c1Chart1.View.AxisX.MajorTickThickness = 3;
c1Chart1.View.AxisX.MajorTickHeight = 10;
c1Chart1.View.AxisX.MajorTickOverlap = 0;
c1Chart1.View.AxisY.MajorGridStrokeThickness = 0;
c1Chart1.View.AxisY.MajorTickThickness = 3;
c1Chart1.View.AxisY.MajorTickHeight = 10;
c1Chart1.View.AxisY.MajorTickOverlap = 0;
```

下面的图像显示MajorTickOverlap 属性的值为 0 时的结果：



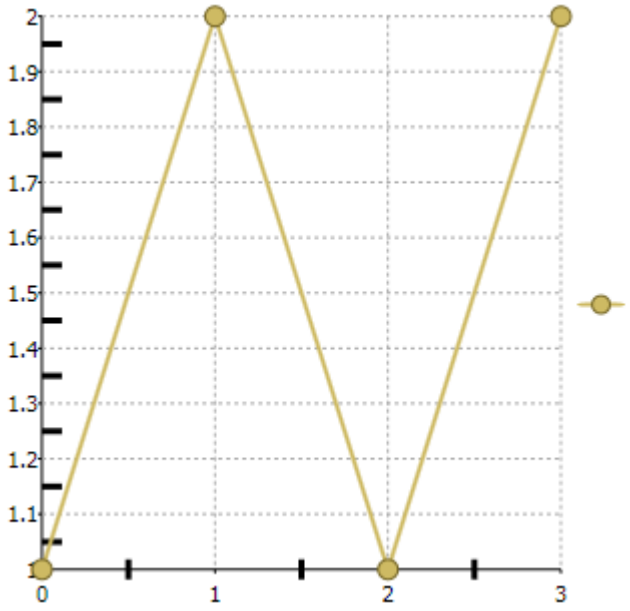
次要刻度线重叠

您可以通过为MinorTickOverlap （在线文档 'MinorTickOverlap 属性'）属性指定一个从 0 到 1 的值，决定次要刻度线标记的重叠程度。默认值是0，这意味着没有重叠。当重叠是1，整个刻度线位于绘图区域内。

C#

```
c1Chart1.Reset(true);

c1Chart1.Data.Children.Add(
new DataSeries() { ValuesSource = new double[] { 1, 2, 1, 2 } });

c1Chart1.ChartType = ChartType.LineSymbols;
c1Chart1.View.AxisX.MinorGridStrokeThickness = 0;
c1Chart1.View.AxisX.MinorTickThickness = 3;
c1Chart1.View.AxisX.MinorTickHeight = 10;
c1Chart1.View.AxisX.MinorTickOverlap = .5;

c1Chart1.View.AxisY.MinorGridStrokeThickness = 0;
c1Chart1.View.AxisY.MinorTickThickness = 3;
c1Chart1.View.AxisY.MinorTickHeight = 10;
c1Chart1.View.AxisY.MinorTickOverlap = 1;
```

下图描述了一个MinorTickOverlap 设置为“1”时的结果：



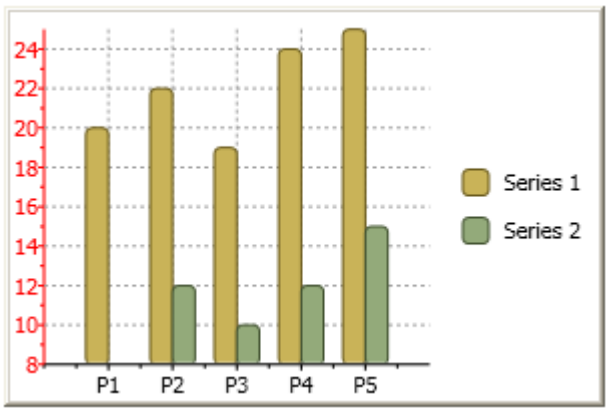
通过代码指定主次刻度线

坐标轴上具有两种不同类型的刻度线：主刻度线具有一条短线以及相关联的标签，而次刻度线仅具有一条和坐标轴相交的短线。
默认情况下，刻度线之间的间距是自动计算的。

为了设置一个特定的距离，可以使用MajorUnit（在线文档 'MajorUnit 属性'）以及MinorUnit（在线文档 'MinorUnit 属性'）属性。

默认刻度线

下图显示默认的刻度线：



自定义刻度线

下图显示了使用MajorUnit以及MinorUnit属性设置特定的距离的图表，例如：

Visual Basic

```
c1Chart1.View.AxisY.MajorUnit = 5
```

```
c1Chart1.View.AxisY.MinorUnit = 1
```

C#

```
c1Chart1.View.AxisY.MajorUnit = 5;
c1Chart1.View.AxisY.MinorUnit = 1;
```

时间坐标轴

对于时间坐标轴您可以指定MajorUnit以及MinorUnit为一个TimeSpan值：

Visual Basic

```
c1Chart1.View.AxisY.MajorUnit = TimeSpan.FromHours(12)
```

C#

```
c1Chart1.View.AxisY.MajorUnit = TimeSpan.FromHours(12);
```

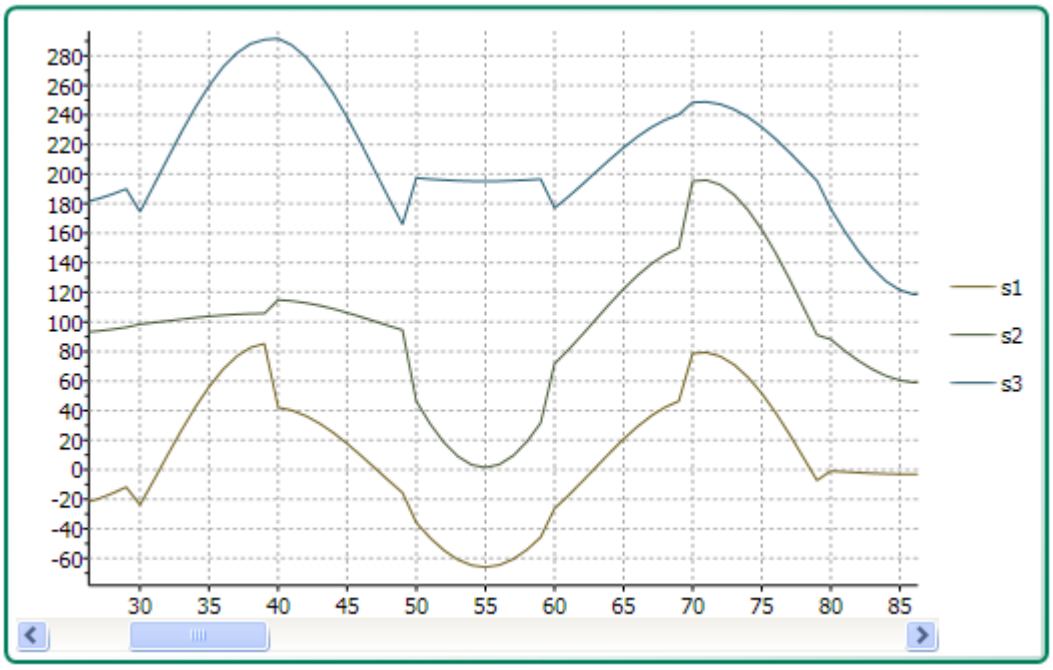
网格线

网格线是在单位主/次的间隔中垂直于主要/次要刻度线的线。当您在试图精确定位图表中的某个值时，网格线可以帮助提高图表的可读性。

- **绘制或填充主要/次要网格线**
  您可以通过MajorGridStroke（在线文档 'MajorGridStroke 属性'）/MinorGridStroke（在线文档 'MinorGridStroke 属性'）属性
  应用颜色至主次要网格线在每一个主网格线的值之间，可以通过MajorGridFill（在线文档 'MajorGridFill 属性'）属性应用一个填充色。
- **设置主要/次要网格线的虚线样式**
  你可以通过MajorGridStrokeDashes（在线文档 'MajorGridStrokeDashes 属性'）/MinorGridStrokeDashes（在线文档 'MinorGridStrokeDashes 属性'）属性设置主次网格线的虚线样式。
- **设置主要/次要网格线的线宽**
  您可以通过 MajorGridStrokeThickness（在线文档 'MajorGridStrokeThickness 属性'）/MinorGridStrokeThickness （在线文档 'MinorGridStrokeThickness 属性'）属性指定主/次网格线的线宽。
- **设置主要的网格线的填充**
  您可以使用MajorGridFill （在线文档 'MajorGridFill 属性'）属性为主网格线应用一个填充。

滚动

在您的图表中具有大量的X-值或者Y-值的情况下，您可以向您的图表添加AxisScrollBar（在线文档'AxisScrollBar 类'）至坐标轴。添加滚动条可以使得通过滚动使得您可以一次读取一块，这样可以更加仔细地查看数据。下面的图像显示的是将Scrollbar设置给View.AxisX.Value属性的结果。



滚动条可以出现在X轴或Y轴，仅需要简单地设置ScrollBar的Value属性为AxisX（针对X轴）或者AxisY（针对Y轴）。

下面的XAML代码显示如何为X轴指定水平滚动条：

XAML

```
<c1chart:C1Chart Name="c1Chart1">
    <c1chart:C1Chart.View>
      <c1chart:ChartView>
        <c1chart:ChartView.AxisX>
          <c1chart:Axis Scale="0.2">
            <c1chart:Axis.ScrollBar>
              <c1chart:AxisScrollBar />
            </c1chart:Axis.ScrollBar>
          </c1chart:Axis>
        </c1chart:ChartView.AxisX>
      </c1chart:ChartView>
    </c1chart:C1Chart.View>
```

为滚动条设置最大值和最小值将防止在滚动时，滚动条改变坐标轴的值。

反转和逆向图表坐标轴

当一个数据集包含某个大范围的某个X值或Y值时，有时正常的图表设置并不能显示最有效的信息。格式化一个图表，使其沿着垂直方向显示Y轴，并且坐标轴标注从最小值开始，反转或逆向显示坐标轴有时可以在视觉上更具有吸引力。因此，C1Chart（在线文档'C1Chart 类'）提供了Inverted属性以及Reversed（在线文档 'Reversed 属性'）属性给坐标轴。

设置ChartView（在线文档'ChartView 类'）的Reversed（在线文档 'Reversed 属性'）属性为True将逆向显示坐标轴。这意味着，在坐标轴的最大值一侧将被坐标轴最小值一侧所取代，而最小值一侧也将被最大值一侧取代。最初，该图在X轴的左侧显示的最小值，同时也在Y轴的底部显示最小值。
设置坐标轴的Reversed属性，尽管这将并置Maximum以及Minimum值。

交换X和Y轴

为了在图表加载之后反转坐标轴，请使用以下代码：

C#

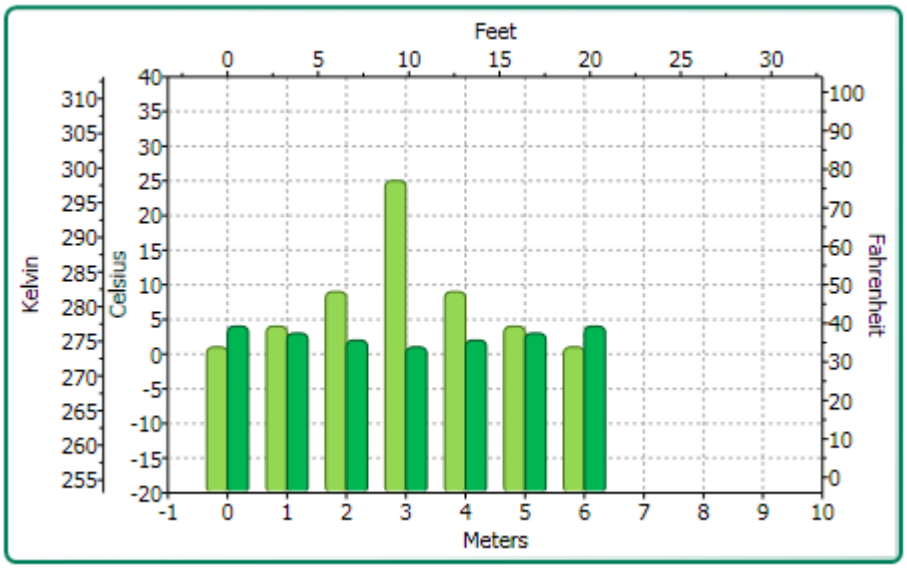
```
c1Chart1.View.Inverted = true;
```

多坐标轴显示

当您存在以下场景时，通常会需要用到多重坐标轴：

- 两个或更多地具有混合数据类型的数据系列，
- 使得数据范围在很宽的范围，且不同的数据系列范围不一致。

下面的图表用五个坐标轴，有效地使用米制和非米制测量单位标识长度和温度：



您可以通过添加一个新的Axis对象，并通过AxisType（在线文档 'AxisType 属性'）属性指定其类型（X，Y或Z轴）以添加多个坐标轴至图表。

下面的XAML标记显示如何添加多个Y轴至图表：

XAML

```
<c1chart:C1Chart Margin="0" Name="c1Chart1">
  <c1chart:C1Chart.View>
    <c1chart:ChartView>
      <!--辅助 Y 轴 -->
      <c1chart:Axis Name="ay2" AxisType="Y" Position="Far" Min="0" Max="10" />
      <c1chart:Axis Name="ay3" AxisType="Y" Position="Far" Min="0" Max="20" />
      <c1chart:Axis Name="ay4" AxisType="Y" Position="Far" Min="0" Max="50" />
    </c1chart:ChartView>
  </c1chart:C1Chart.View>
  <c1chart:C1Chart.Data>
```

```
    <c1chart:ChartData>
      <c1chart:DataSeries Values="1 2 3 4 5" />
      <c1chart:DataSeries AxisY="ay2" Values="1 2 3 4 5" />
      <c1chart:DataSeries AxisY="ay3" Values="1 2 3 4 5" />
      <c1chart:DataSeries AxisY="ay4" Values="1 2 3 4 5" />
    </c1chart:ChartData>
  </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

缩放多个坐标轴

为了缩放每一个独立的坐标轴，您应当将每一个坐标轴的scale和value属性在PropertyChanged事件中关联，如以下代码所示：

C#

```
//假定 ay2 是辅助的Y轴

 ((INotifyPropertyChanged)chart.View.AxisY).PropertyChanged += (s, e) =>
   {
     if (e.PropertyName == "Scale")
     {
       ay2.Scale = chart.View.AxisY.Scale;
     }
     else if (e.PropertyName == "Value")
     {
       ay2.Value = chart.View.AxisY.Value;
     }
   };
```

对数标度

当数据显示具有较大的范围差异，或者当数据预期在同一张图表中成指数关系变化，则通常使用对数缩放一个或多个坐标轴将会比较方便。在对数坐标轴上，沿着它描绘了一个相等的百分比变化。如果在一个或两个坐标轴上使用对数缩放，则该图称为对数范围图表。

与对数缩放，值是基于物理上的间隔的值，而不是值本身的对数。当数量的数据在图表中展示的范围非常宽，以及期望几何和/或指数关系描述时非常有用。

与算术图表类型采用直接衡量的单位不同，对数缩放图表展示变化的百分比变化。例如，在对数尺度图中测量美元，改变从1美元至2美元是百分之百的改变所以距离图轴从$ 1到2美元的变化和从50美元到100美元的变化是一样的。而在一个算术图表中，从50美元到100美元的变化将使距离轴上从50美元到100美元出现较大的图表，因为这是一个改变50美元而不是仅为1美元。

常用对数

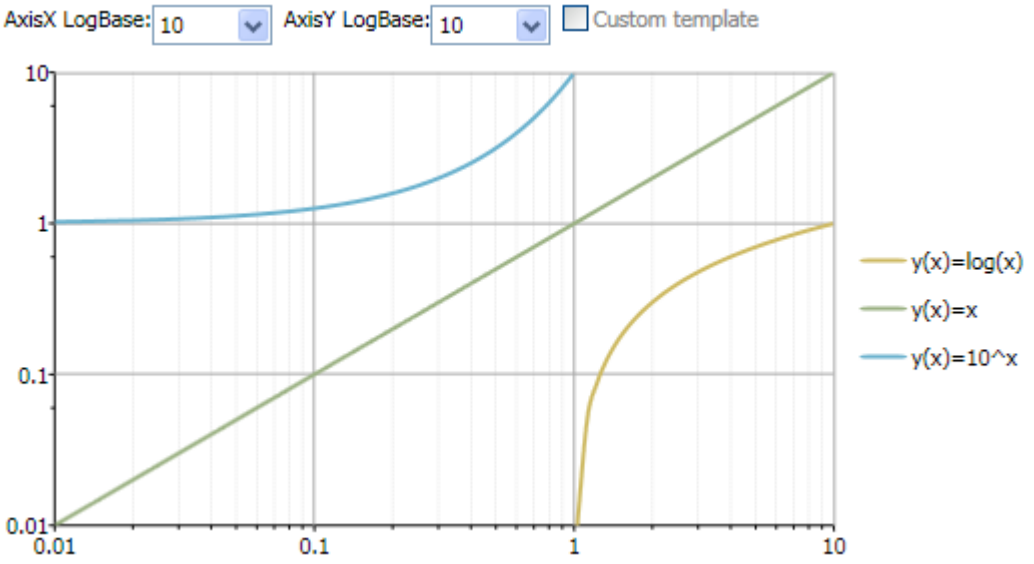
对数可以使用任何基值表示，包括整数和浮点值。最常用的两种类型的数据包括：

- 常用对数 –采用 10 作为基数，因此 log 100 = 2。
- 自然对数–使用数学常数e为基。

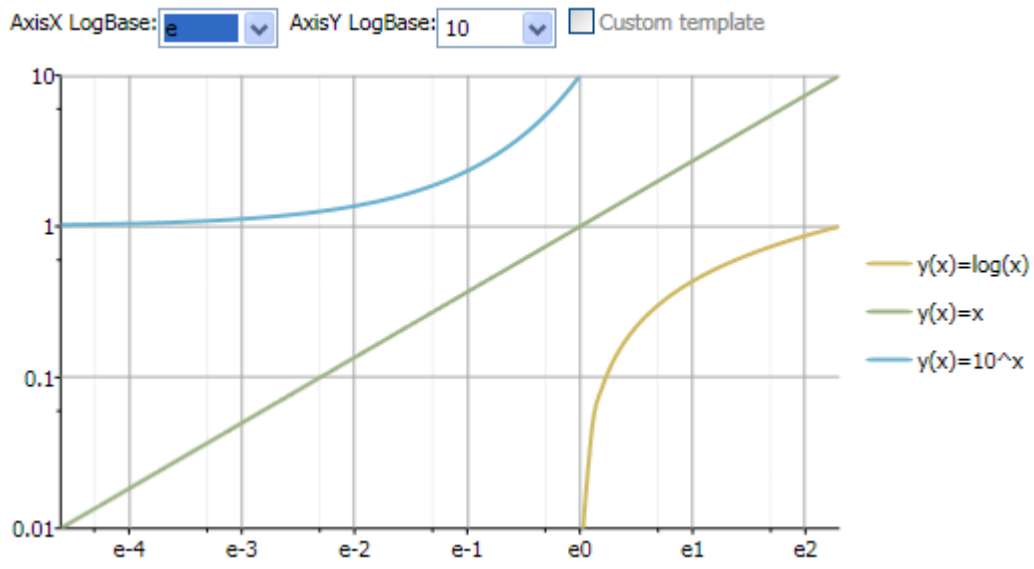
对数基数

您可以通过LogBase（在线文档 'LogBase 属性'）属性指定对数的基数。默认值是double.NaN，对应于默认的线性轴。自然对数是以常数e为基数的对数。请注意，当值小于或等于零的时候，对数比例具有数学意义。

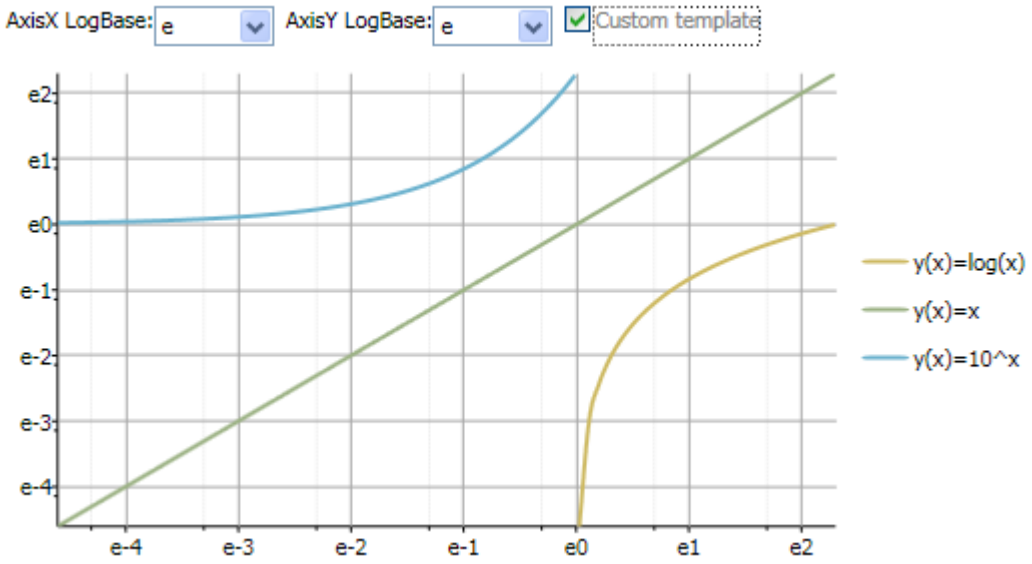
下图显示了当X-轴和Y-轴的LogBase设置为10，即常用对数时的图像：



下图显示了当X-轴的LogBase为e，而Y-轴的LogBase设置为10时的图像：



下图显示了当X-轴的LogBase为e，同时Y-轴的LogBase设置也为e时的图像：



对数坐标轴必须遵循下列附加标准：

- 任何小于或者等于零的数值将不产生图像（作为数据空白点处理），这是由于对数坐标轴仅能够处理大于零的数据值。基于同样的原因，坐标轴和数据的最小/最大范围和原点属性不能被设置为零或更少。
- 坐标轴的numbering增量，刻度增量，以及精度属性当坐标轴为对数轴时不起作用。
- 对于一个对数的X-轴，图表的类型必须是散点图、气泡图、面积图、HiLo，HiLoOpenClose或者蜡烛图中的一种。对于Y-轴，图表的类型必须是散点图，气泡图，面积图，极坐标图，HiLo，HiLoOpenClose，蜡烛图，雷达或填充雷达图中的一种。

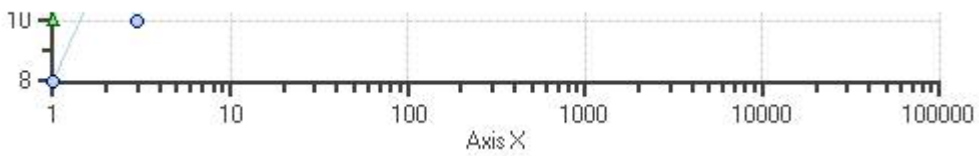
UnitMajor 和对数坐标轴

对于对数轴缩放，MajorUnit（在线文档 'MajorUnit属性'）作为一个乘数的每个周期，提供了一个提示注释间距在每个周期内的对数的底基值。即（MajorUnit *基周期值）大约是每个周期内的注释增值。对于整数的对数基准值，结果通常是准确的。对于浮点值，注释是以很好的数字为线性缩放的。

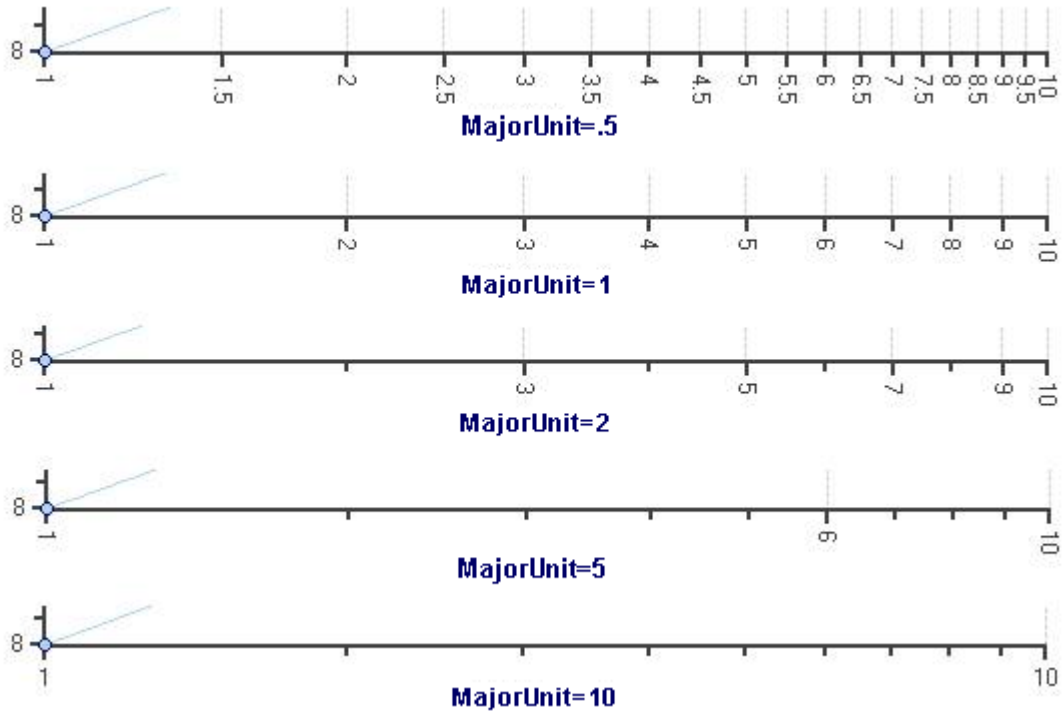
关于UnitMajor以及Logarithmic坐标轴的详细解释

通常，当使用对数刻度时，一个图表轴的范围将跨越多个周期的对数基数。在这些情况下，对MajorUnit来说通常的线性规范不再是有道理的，作为一个适当的值的一个给定的周期为上一个或下一个周期是没有意义的。对于MajorUnit设置的值而言，它必须是相对于对数基数的周期。

如果这对你没有意义，要想一想，固定的，增量的值，你可以使用以下的轴：



根据上面的推理，在对数坐标轴的图，假设MajorUnit 指定每个周期的基础值的分数。考虑下面的例子：



在每一种情况下，基准周期值为1。每个周期的下一个标注值=前一个数+（基数周期* MajorUnit）。MajorUnit 的最大值是LogarithmicBase。MajorUnit的自动值始终是LogBase。

当所有的注释值被计算，一个很好的四舍五入算法将应用到数值，使得该数值容易被阅读。该行为可能有点奇怪，但它是可容纳任何对数的基数，同时获取的标注的数值也是容易阅读的。

例如，以上绘图区是以10为底的常用对数，但是同样也有固有的其他底数值，比如说以2为底，以x为底等等。

图表图例

隐藏图表图例

以编程方式隐藏图表图例可以参考以下：

在XAML中向图例添加名称，之后您可以通过代码修改其可见性：legend.Visibility = ...

XAML

```
<c1chart:C1Chart x:Name="chart" >
    <c1chart:C1ChartLegend x:Name="legend"  />
    ...
 </c1chart:C1Chart>
```

改变图例的方向和位置

为了在图表控件底部居中、水平显示图表图例，请使用以下代码：

C#

```
C1ChartLegend.Orientation = Horizontal;
C1ChartLegend.Position = C1.WPF.C1Chart.LegendPosition.BottomCenter;
```

图表视图

ChartView（在线文档 'ChartView 类'）对象表示图表中包含数据的区域（包括标题和图例，但是不包括坐标轴）。View属性返回一个ChartView对象。

设置绘图区背景

设置绘图区背景是一种比较简单的方式添加一些自定义以及提高您图表的对比度。
下面的XAML标记允许您设置背景：

XAML

```
<c1:C1Chart.View>
    <c1:ChartView PlotBackground="#FF343434">
    </c1:ChartView>
</c1:C1Chart.View>
```

你也可以在代码中设置PlotBackground 属性。只需要给您的图表指定名字，然后将以下代码添加到InitializeComponent()方法中：

C#

```
chart.View.PlotBackground = Brushes.BlueViolet;
```

数据绑定

WPF及Silverlight版Chart中的C1Chart（在线文档 'C1Chart 类'）控件可以绑定到任意实现了System.Collections.IEnumerable的对象（比如说XmlDataProvider、DataSet、DataView、等等）。

一个数据表可以通过指定给C1Chart控件的ItemsSource属性绑定到图表。

下面的主题提供不同的数据绑定方法用于将数据发送到C1Chart控件的信息。

值的集合

你可以用几种方法将数据传送到图表中。一种方法是通过ValuesSource（在线文档 'ValuesSource 属性'）属性绑定一个值的集合。

任何支持IEnumerable接口的数值型值的集合可以用作数据系列的数据源。每个数据序列类型均具有适当的属性用作数据绑定。例如，DataSeries（在线文档 'DataSeries 类'）类使用ValuesSource属性用做数据绑定。

为了绑定数值的集合至DataSeries，您首先应当指定该绑定源作为一组double值类型的数组，如以下所示：

XAML

```
<!—绑定源 -->
```

```
<x:Array xmlns:sys="clr-namespace:System;assembly=mscorlib"
       x:Key="array" Type="sys:Double">
       <sys:Double>1</sys:Double>
       <sys:Double>4</sys:Double>
       <sys:Double>9</sys:Double>
       <sys:Double>16</sys:Double>
</x:Array>
```

将数组传递给数据序列，使用下面的标记：

XAML

```
<!–绑定目标 -->
<c1chart:C1Chart Name="chart">
  <c1chart:C1Chart.Data>
     <c1chart:ChartData ItemsSource="{Binding Source={StaticResource array},
Path=Items}">
       <c1chart:DataSeries ValuesSource="{Binding Source={StaticResource
array},Path=Items}"/>
     </c1chart:ChartData>
   </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

可以将数据值作为属性指定，这组值应当使用空格分隔，例如：

XAML

```
<c1chart:DataSeries Values="1 2 9 16"/>
```

之前的标记声明绑定DataSeries的ValuesSource属性至DataSeries对象的Items属性，给定了一个值"1 2 9 16"。

对象集合

当你有一个集合的对象，每个对象包括数字属性时，应该使用数据绑定。数据绑定过程中至少有两个图表属性参与这一过程。

- ItemsSource属性 - 该对象集合分配到的源。
- ValueBinding属性 – 获取或设置图表的数据系列的值绑定。指定对象的哪一个属性提供数据值。

假设在资源中我们有一个点的数组。

XAML

```
x:Array x:Key="points" Type="Point">
       <Point>0,0</Point>
       <Point>10,0</Point>
       <Point>10,10</Point>
       <Point>0,10</Point>
       <Point>5,5</Point>
</x:Array>
```

以下XAML代码片段表示一个具有两个数据系列的图表，一个绑定到点的X坐标，另一个绑定到该点的Y坐标：

XAML

```
<c1chart:C1Chart Name="chart2">
       <c1chart:C1Chart.Data>
              <c1chart:ChartData
 ItemsSource="{Binding Source={StaticResource points}, Path=Items}">
                     <c1chart:DataSeries ValueBinding="{Binding Path=X}"/>
                     <c1chart:DataSeries ValueBinding="{Binding Path=Y}"/>
              </c1chart:ChartData>
       </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

下一个示例展示系列同时使用点的两个坐标值；请注意这里是XYDataSeries（在线文档 'XYDataSeries 类'）类处理两组分别关联到 X-轴和Y-轴坐标值的数据值。

XAML

```
<c1chart:XYDataSeries
       XValueBinding="{Binding Path=X}"
       ValueBinding="{Binding Path=Y}"/>
```

Observable集合

WPF和Silverlight具有一个特殊的泛型集合类型，ObservableCollection，该类提供了关于更新的各种通知，比如说当添加项目，移除项目，或者整个列表刷新。如果使用该类的实例作为图表的数据源，则该图将自动反映集合中所做的更改。

在代码中，添加System.Collections.ObjectModel命名空间至您的页面（以及C1.WPF.C1Chart 或者C1.Silverlight.Chart）。这包括ObservableCollection。

C#

```
using System.Collections.ObjectModel;
using C1.WPF.C1Chart;
//或者
using C1.Silverlight.Chart;
```

然后声明一个Point类型的ObservableCollection集合。这将做为我们的图表的数据来源：

C#

```
ObservableCollection<Point> points = new ObservableCollection<Point>();
```

清除所有预设图表数据（如果有的话），然后用一些虚拟值填充点集合。

C#

```
//清除图表数据
c1Chart1.Data.Children.Clear();

//创建虚拟数据
points.Add(new Point(0, 20));
```

```
points.Add(new Point(1, 22));
points.Add(new Point(2, 19));
points.Add(new Point(3, 24));
points.Add(new Point(4, 29));
points.Add(new Point(5, 7));
points.Add(new Point(6, 12));
points.Add(new Point(7, 15));
```

Next, create a XYDataSeries bound to this collection and add it to the chart.

C#

```
//设置C1Chart 数据系列
XYDataSeries ds = new XYDataSeries();
ds.Label = "Series 1";
//绑定数据系列至集合。
ds.ItemsSource = points;
//重要：当使用ItemsSource时设置绑定
ds.ValueBinding = new Binding("Y");
ds.XValueBinding = new Binding("X");
//添加数据系列至图表
c1Chart1.Data.Children.Add(ds);
```

您可以直接将此点的集合绑定到数据系列的ItemsSource上。同样重要的是要指定ValueBinding （Y）和XValueBinding至Point对象的X和Y字段。就像如果这是您的自定义业务对象，您必须将数据系列的值绑定到所需的字段。然后将数据序列添加到图表的数据集合中。您可以很容易地通过这种方法添加多个数据序列。

数据上下文绑定

如果你计划有多个属性绑定到同一个源，那么您可以考虑使用DataContext属性。DataContext属性提供了一种方便的方式来建立一个数据的作用域。

当其ItemsSource未设置时，C1Chart控件将使用DataContext属性作为ItemsSource。和ItemsSource一样，DataContext应当是IEnumerable类型。

双精度数组的数据上下文

下面的代码演示了如何使用双精度数组作为数据上下文：

C#

```
c1Chart1.Reset(true);
c1Chart1.DataContext = new double[] { 1, 2, 3, 4, 5 };
c1Chart1.ChartType = ChartType.Column;
```

点数组的数据上下文

下面的代码演示了如何使用点的数组作为数据上下文：

C#

```
c1Chart1.Reset(true);
c1Chart1.DataContext = new Point[] { new Point(1, 1), new Point(2, 2), new Point(3,
4), new Point(4, 1) };
c1Chart1.ChartType = ChartType.LineSymbols;
```

数据系列绑定

C1Chart 提供以下绑定类型用于指定哪些属性应该被绘制在图表上：

- 项名称绑定-指定图表数据的项目名称。
- 系列绑定-在自动生成过程中生成的数据序列中的每个绑定的数据系列的值绑定集合。
- X值绑定 - X值绑定指定图表数据系列绑定值。

项目名称绑定

项目名称绑定是一个数据绑定类型，用于指定当使用了ItemNameBinding（在线文档 'ItemNameBinding 属性'）属性时，图表数据的项目名称绑定。

下面的示例演示调用目标对象的绑定方法：

C#

```
ChartBindings bindings = new ChartBindings();
bindings.ItemNameBinding = new Binding("Name");
bindings.SeriesBindings.Add(new Binding("Input"));
bindings.SeriesBindings.Add(new Binding("Output"));

chart.Bindings = bindings;
chart.DataContext = new InOut[]
{
    new InOut() { Name = "n1", Input = 90, Output = 110},
    new InOut() { Name = "n2", Input = 80, Output = 70},
    new InOut() { Name = "n3", Input = 100, Output = 100},
};

在此，InOut 定义为：

public class InOut
{
    public string Name { get; set; }
    public double Input{ get; set;}
    public double Output { get; set; }
}
```

X值绑定

当使用了XBinding 属性时，X-值绑定指定图表数据的 x-值绑定。

下面的例子使用了XBinding 属性为数据系列设置x-值绑定：

C#

```
ChartBindings bindings = new ChartBindings();
bindings.XBinding = new Binding("X");
bindings.SeriesBindings.Add(new Binding("Y"));
chart.Bindings = bindings;
chart.DataContext = new Point[] { new Point(1, 0),
new Point(2, 2), new Point(3, 1), new Point(5, 3) };
```

从DataSet中绑定DataTable

这里是从DataTable创建图表的示例代码。

Visual Basic

```
Private _dataSet As DataSet
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' 创建连接并填充数据集
    Dim mdbFile As String = "c:\db\nwind.mdb"
    Dim connString As String = String.Format("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source={0}", mdbFile)
    Dim conn As New OleDbConnection(connString)
    Dim adapter As New OleDbDataAdapter("SELECT TOP 10 ProductName, UnitPrice FROM
Products " & vbCr & vbLf & " ORDER BY UnitPrice;", conn)
    _dataSet = New DataSet()
    adapter.Fill(_dataSet, "Products")
    ' 将数据表列设置为图表数据的数据源
    c1Chart1.Data.ItemsSource = _dataSet.Tables("Products").Rows
End Sub
```

C#

```
DataSet _dataSet;
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  // 创建连接并填充数据集
  string mdbFile = @"c:\db\nwind.mdb";
  string connString = string.Format(
    "Provider=Microsoft.Jet.OLEDB.4.0; Data Source={0}",
    mdbFile);
  OleDbConnection conn = new OleDbConnection(connString);
  OleDbDataAdapter adapter = new OleDbDataAdapter(
     @"SELECT TOP 10 ProductName, UnitPrice FROM Products
```

```
        ORDER BY UnitPrice;", conn);
  _dataSet = new DataSet();
  adapter.Fill(_dataSet, "Products");
  // 将数据表列设置为图表数据的数据源
  c1Chart1.Data.ItemsSource = _dataSet.Tables["Products"].Rows;
}
```

XAML

XAML

```
<c1chart:C1Chart.Data>
  <c1chart:ChartData ItemNameBinding="{Binding Path=[ProductName]}">
    <c1chart:DataSeries ValueBinding="{Binding Path=[UnitPrice]}"/>
  </c1chart:ChartData>
</c1chart:C1Chart.Data>
```

数据点转换器

在XAML中创建复杂点标签的模版时，DataPointConverter类将非常有用。
DataPointConverter 使用 converter参数基于当前数据点属性生成字符串。转换器参数字符串可以包含以下关键字，每一个关键字可以被每一个数据点的属性的真实值替换：

- #Values -数据点的 Y - 坐标值。
- #XValues -数据点的X-坐标值（对于XYDataSeries）。
- #PointIndex -数据点的索引。
- #SeriesIndex -数据系列的索引。
- #SeriesLabel -数据系列标签。
- #NewLine -新行。

关键词从#开始，并且应当包括在一对花括号中。可选的格式字符串可以添加到括号内部，如以下字符串格式：{#Values:0.0}

下面的XAML标记显示如何使用DataPointConverter 类：

XAML

```
<c1chart:C1Chart Name="chart" ChartType="LineSymbols" Margin="20" >
   <c1chart:C1Chart.Resources>
     <c1chart:DataPointConverter x:Key="cnv"/>
   </c1chart:C1Chart.Resources>
   <c1chart:C1Chart.Data>
     <c1chart:ChartData>
       <c1chart:XYDataSeries Label="S1"
         XValues="1,2,3,4,5,6,7" Values="1,2,3,4,3,4,2" >
         <c1chart:XYDataSeries.PointLabelTemplate>
           <DataTemplate>
             <Border BorderBrush="Black" BorderThickness="0.5"
```

```
                Background="#70FFFFFF"
                c1chart:PlotElement.LabelAlignment="MiddleCenter">
                <TextBlock>
                  <TextBlock.Text>
                    <Binding Converter="{StaticResource cnv}">
                      <Binding.ConverterParameter>
                        {#SeriesLabel}{#NewLine}
                        X={#XValues:0.0},Y={#Values:0.0}{#NewLine}
                     SI={#SeriesIndex},PI={#PointIndex}
                      </Binding.ConverterParameter>
                    </Binding>
                  </TextBlock.Text>
                </TextBlock>
              </Border>
            </DataTemplate>
          </c1chart:XYDataSeries.PointLabelTemplate>
        </c1chart:XYDataSeries>
      </c1chart:ChartData>
   </c1chart:C1Chart.Data>
</c1chart:C1Chart>
```

数据标签

数据标签是图表中关联到数据点的标签。它们在某些类型的图表上非常有用，可以使得更加容易地看到某个特定的数据点属于某一个系列或者该数据点的精确值。

C1Chart 支持数据标签。每个数据系列都有一个PointLabelTemplate（在线文档 'PointLabelTemplate 属性'）属性，该属性指定显示在每一个数据点附近的视觉元素。PointLabelTemplate 通常作为一个XAML资源定义，可以通过XAML或代码指定给图表。

您可以添加一个DataTemplate以同时决定数据如何展示以及数据绑定如何访问现有数据。

为定义PointLabelTemplate 为XAML资源，您可以创建一个资源字典，添加DataTemplate 资源至您的资源字典，之后再您的Window.xaml文件中，您可以访问该DataTemplate资源。

添加新资源字典：

1. 在“解决方案资源管理器”中，右键单击“项目”，指向“添加”，然后选择“资源字典”。将出现“添加新项”对话框。
2. 在“名称”文本框中，将字典命名为resources.xaml，并点击“添加”按钮。
3. Resources.xaml添加到项目并在代码编辑器中打开。

为创建一个标签，您需要创建标签模版并将PointLabelTemplate 设置给模版。

当绘制每个数据点的图时，将根据指定的模板创建标签。标签的DataContext属性设置为当前数据点的实例，提供点信息。当使用数据绑定时，它会更容易在标签中显示该信息。

这里是一个标签模板的示例，该模板显示该点的值。

XAML

```
<DataTemplate x:Key="lbl">
   <TextBlock Text="{Binding Path=Value}" />
</DataTemplate>
```

定义了资源之后，可以将资源用于通过使用资源标记扩展语法来定义一个属性值，它指定了key的名称。

为了指定模版至数据系列，需要设置PointLabelTemplate 属性，如下所示：

XAML

```
<c1chart:DataSeries PointLabelTemplate="{StaticResource lbl}" />
```

因为它是一个标准的数据模板，可以建立复杂的标签，例如，下一个样品模板定义为XY图表显示数据点坐标数据标签。

它使用标准的Grid，以两列和两行作为容器。该点的x-值由DataPint类的索引获取。索引器允许为支持多个数据集的数据系列类，比如说XYDataSeries类，获取数值。

XAML

```
<DataTemplate x:Key="lbl">
   <!-- Grid 2x2 with black border  -->
   <Border BorderBrush="Black">
          <Grid>
                 <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                 </Grid.ColumnDefinitions>
                 <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                 </Grid.RowDefinitions>
                 <!-- x-coordinate -->
                 <TextBlock Text="X=" />
                 <TextBlock Grid.Column="1" Text="{Binding Path=[XValues]}" />
                 <!-- y-coordinate -->
                 <TextBlock Grid.Row="1" Text="Y=" />
                 <TextBlock Grid.Row="1" Grid.Column="1" Text="{Binding Path=Value}"
/>
          </Grid>
   </Border>
</DataTemplate>
```

当显示数值数据值时，通常需要格式化输出值。通过静态类Format（在线文档'Format属性'）您可以在XAML代码内部指定标准.NET格式化字符串。例如，示例代码使用转换器来格式化百分比值。

XAML

```
<DataTemplate x:Key="lbl1">
   <TextBlock Text="{Binding Path=PercentageSeries,
          Converter={x:Static c1chart:Converters.Format},
          ConverterParameter=#.#%}"/>
</DataTemplate>
```

数据系列

C1Chart中的重要属性之一就是数据系列。数据系列包含了图表中所包含的所有数据，以及许多重要的数据相关属性。

图表数据系列类型

C1Chart 提供以下数据系列类型，以便有效地处理不同的数据类型：

- BubbleSeries
- DataSeries
- HighLowOpenCloseSeries
- HighLowSeries
- XYDataSeries
- XYZDataSeries

位于DataSeries 类的Label属性表示显示在图表图例区的系列名称的标签。

有几个DataSeries类从相同的DataSeries基类型派生，每一种组合了几种不同的数据集，取决于合适的数据特性。例如，在XYDataSeries 提供两组数据值对应的x和y坐标。在下表中列出了可用的数据序列类的列表。

类	值属性	值绑定属性
DataSeries	Values, ValuesSource	ValueBinding
XYDataSeries	Values, ValuesSource XValues, XValuesSource	ValueBinding XValueBinding
HighLowSeries	Values, ValuesSource XValues, XValuesSource HighValues, HighLowSeries.HighValuesSource LowValues, HighLowSeries.LowValuesSource	ValueBinding XValueBinding HighValueBinding LowValueBinding
HighLowOpenCloseSeries	Values, ValuesSource XValues, XValuesSource HighValues, HighLowSeries.HighValuesSource LowValues, HighLowSeries.LowValuesSource OpenValues, HighLowOpenCloseSeries.OpenValuesSource CloseValues, HighLowOpenCloseSeries.CloseValuesSource	ValueBinding XValueBinding HighValueBinding LowValueBinding OpenValueBinding CloseValueBinding

每一个数据系列类可能具有其默认的用于展示绘图区的模版，例如，HighLowOpenCloseSeries 显示金融数据为一组标记最高值、最低值、开盘价以及收盘价的值的线

图表数据系列外观

每个数据系列的外观由DataSeries（在线文档 'DataSeries 类'）类的三组属性决定：Symbol，Connection，以及ConnectionArea。这些属性会根据图表类型的不同影响图表的不同部分。

Symbol属性确定每个数据点绘制的符号的形状、大小、轮廓和填充属性。它们适用于图表中会显示符号类型，包括折线图、面积图、以及XYPlot图表。Symbol 属性也控制条形图和柱状图中条形的外观。

Connection 属性决定了数据点之间的线条的轮廓和填充属性。它们适用于数据系列的所有点集合。对于折线图，连接线是用来连接点的直线，对于面积图，连接线是包含数据点以下的边框。

DataSeries 和XYDataSeries之间的差异

DataSeries （在线文档 'DataSeries 类'）只有一个逻辑数据值集合（y 值）。

在这种情况下，x值将自动生成（0、1、2.....），同样您也可以通过Data.ItemNames属性指定 x-轴文本标签的值。

XYDataSeries （在线文档 'XYDataSeries 类'）有两组数据的集合，Values（yvalues）以及XValues。

空值

默认情况下，如果在数据值中间存在一个空洞（double.NaN），则图表将仅仅跳过该值并且将直线连接到下一个有效的数据点。

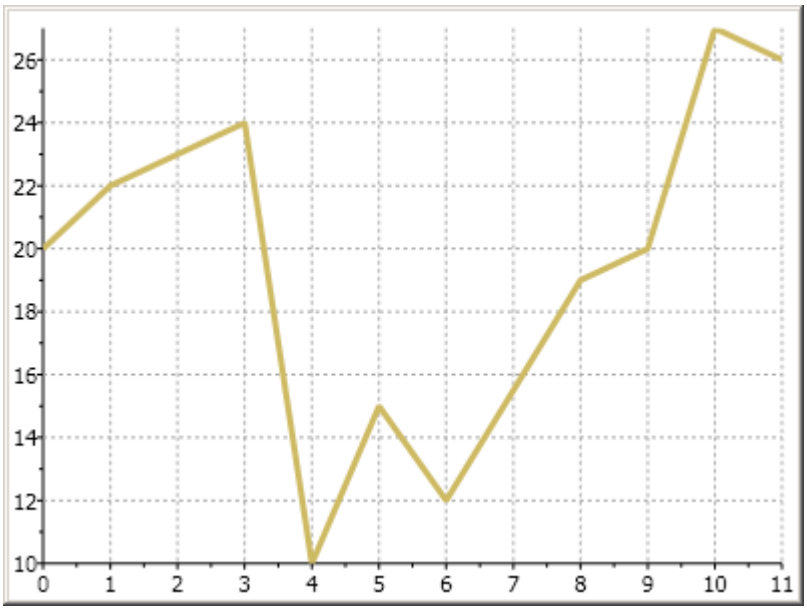
如果要改变这种行为，在存在数据空洞的位置显示一个间隙，则需要设置Display（在线文档 'Display 属性'）=ShowNaNGap。

例如，下面的XAML代码在DataSeries中包含一些特定的数据空洞：

XAML

```
<c1chart:C1Chart  Name="c1Chart1" ChartType="Line">
     <c1chart:C1Chart.Data>
       <c1chart:ChartData>
         <c1chart:DataSeries Values="20 22 NaN 24 15 NaN 27 26"
           ConnectionStrokeThickness="3" />
       </c1chart:ChartData>
     </c1chart:C1Chart.Data>
   </c1chart:C1Chart>
```

当没有设置过Display属性时，图表显示如下图所示：



为了在折线图的图表折线之间显示一个间隙，您可以设置Display属性的值为ShowNaNGap，如下所示：

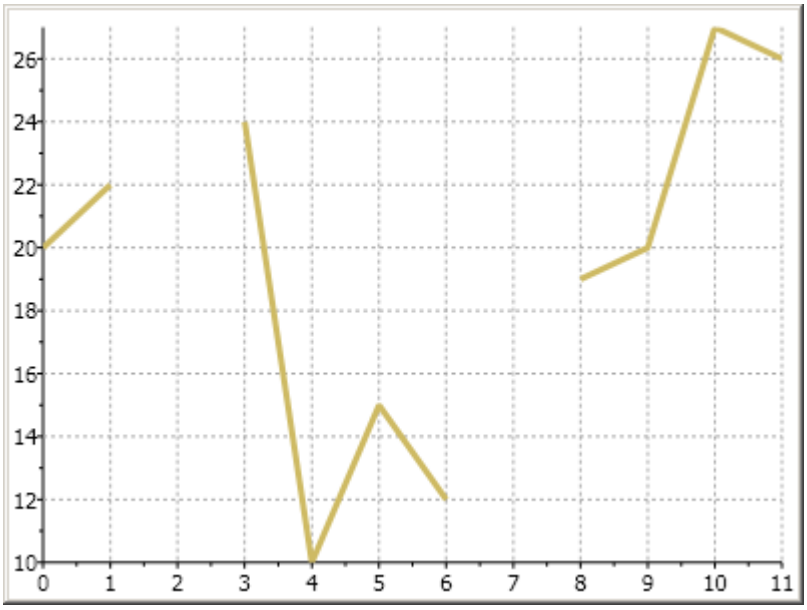
Visual Basic

```
Me.C1Chart1.Data.Children(1).Display = C1.WPF.C1Chart.SeriesDisplay.ShowNaNGap
```

C#

```
this.C1Chart1.Data.Children[1].Display = C1.WPF.C1Chart.SeriesDisplay.ShowNaNGap;
```

线图将显示折线之间的间隙，类似于以下图所示：



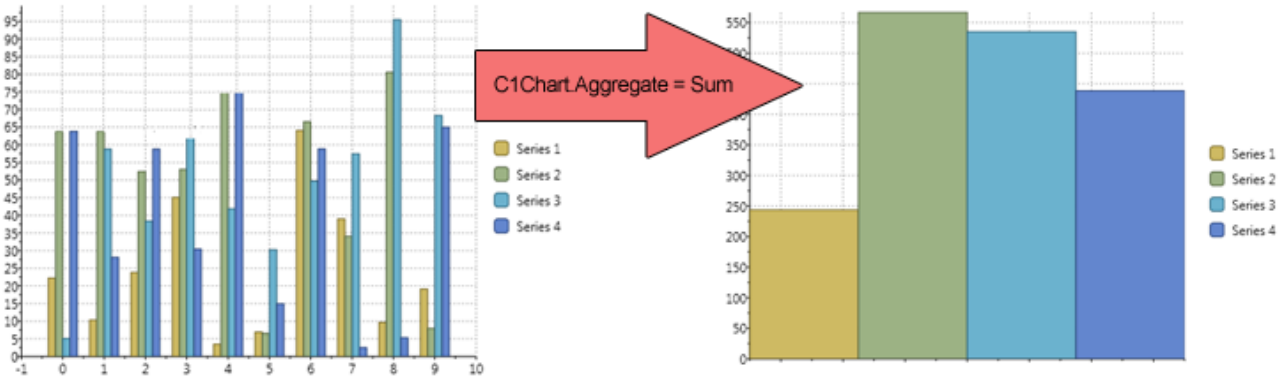
分组和聚合

C1Chart控件支持内置的分组和聚合。这允许你在一个图表中显示汇总数据，而无需做一些将数据分组的额外工作。

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(i) });
}
```

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。

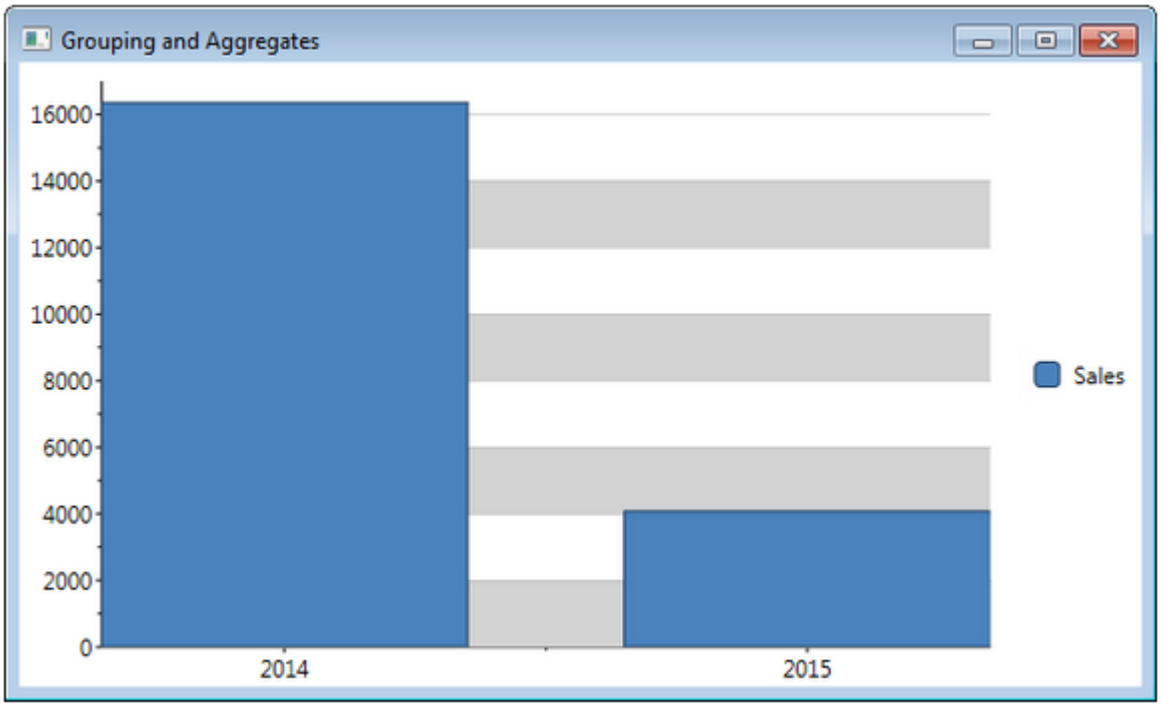
下面的代码定义了GroupSelectorByDate 函数。该函数将调用您的图表中的每个数据点，并确定该数据所属的组。

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#

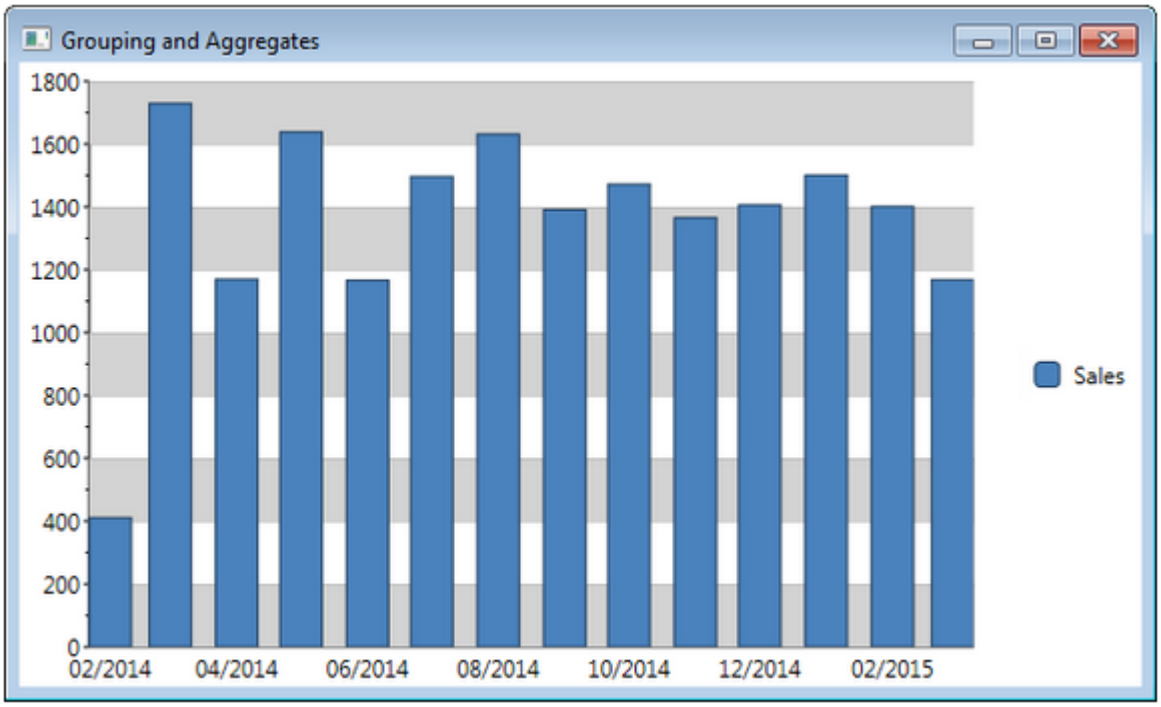
```
double GroupSelectorByDate(double x, double y, object o)
{
    // 返回年份为双精度浮点数
    DateTime dt = x.FromOADate();
    // 为了按照月份进行分组，我们返回该日期的年份和月份值
    return new DateTime(dt.Year, dt.Month, 1).ToOADate();
}
```

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#

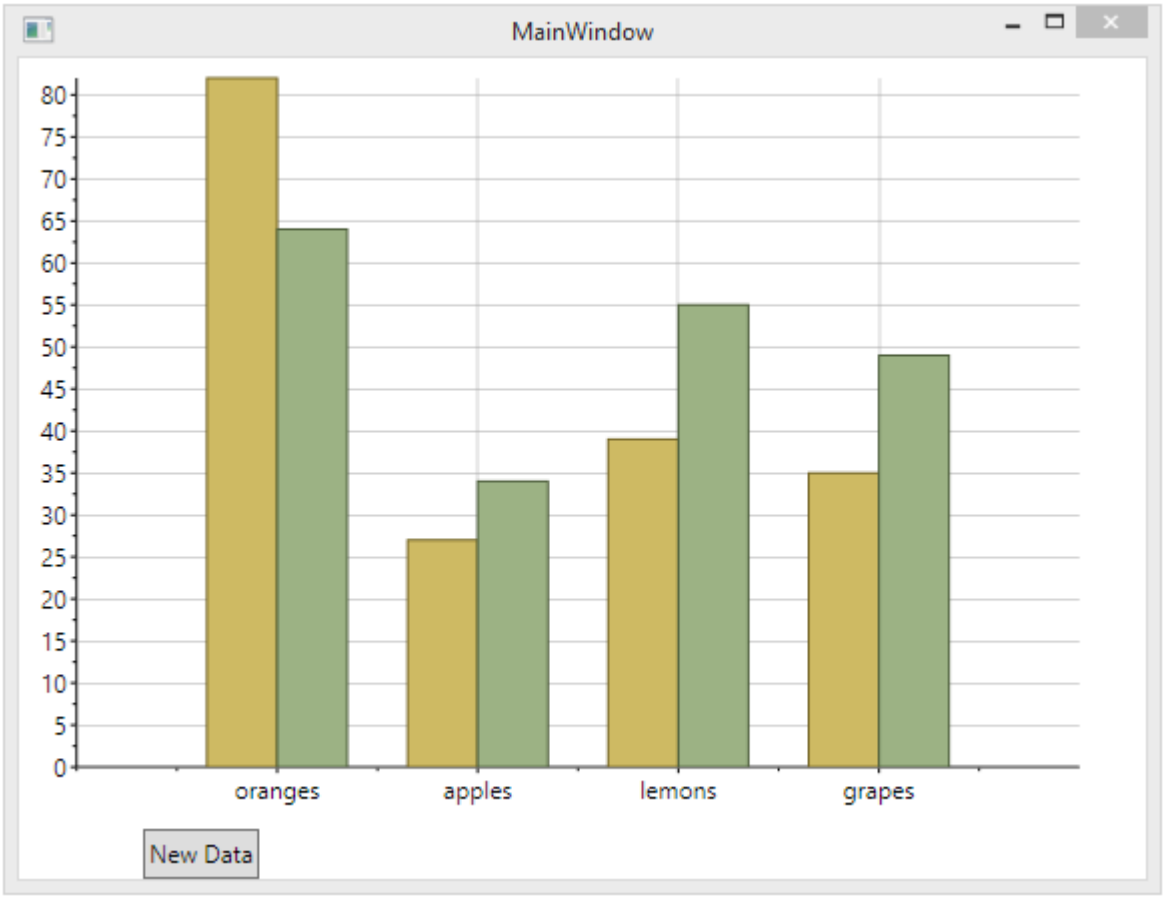
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    chart.Data.Children.Clear();
    CreateSampleChart();
}
```

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
{
    public string Name { get; set; }
    public double Value { get; set; }
    static Random rnd = new Random();
    public static SampleItem[] CreateSampleData(int cnt)
    {
        var names = new string[] { "oranges", "apples", "lemons", "grapes" };
        var array = new SampleItem[cnt];
        for (int i = 0; i < cnt; i++)
        {
            array[i] = new SampleItem() { Value = rnd.Next(1, 10), Name =
names[rnd.Next(names.Length)] };
        }
        return array;
    }
   }
  }
```

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：**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：**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：**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WPF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()
{
XValuesSource = new double[] { 1, 2, 3, 4 },
ValuesSource = new double[] { 1, 2, 3, 4 },
SizeValuesSource = new double[] { 1, 2, 3, 4 },
};

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：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：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

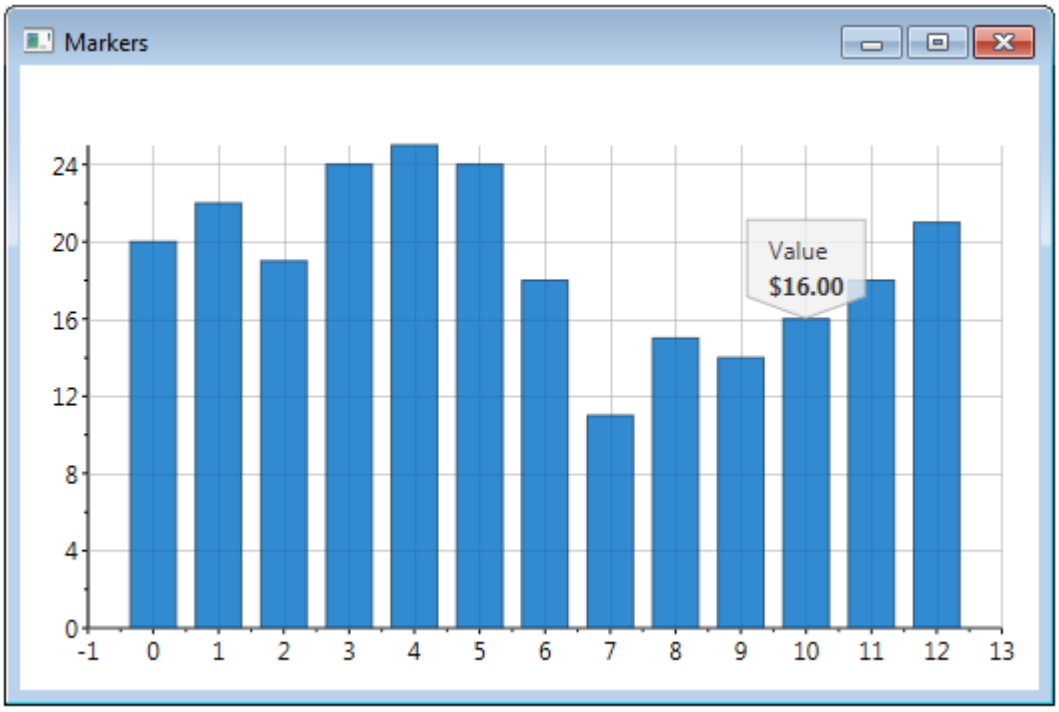
```
<!-- 简单绑定标记 -->
<c1:ChartPanelObject x:Name="obj" Attach="DataX"
                     Action="MouseMove"
                     DataPoint="-1,-1"
                     HorizontalAlignment="Center"
                     VerticalAlignment="Top"
                     Width="60" Height="50">
    <c1:ChartPanelObject.RenderTransform>
        <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：

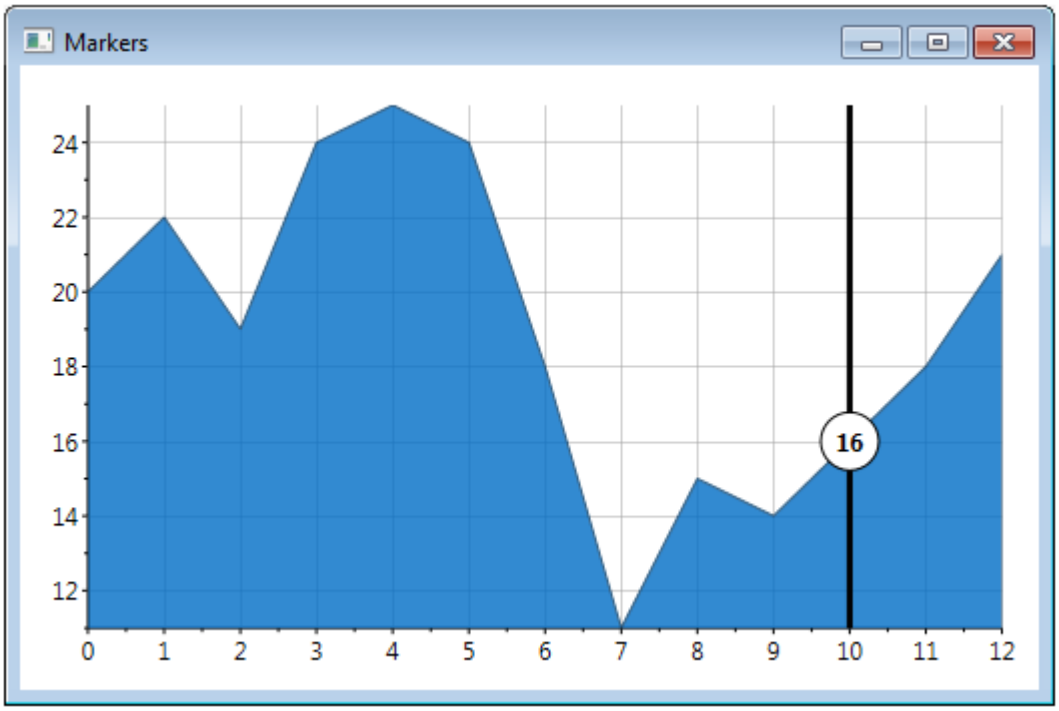
- Attach = DataX
- Action = MouseMove
- DataPoint = -1,-1
- HorizontalAlignment = Center
- VerticalAlignment = Top

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！

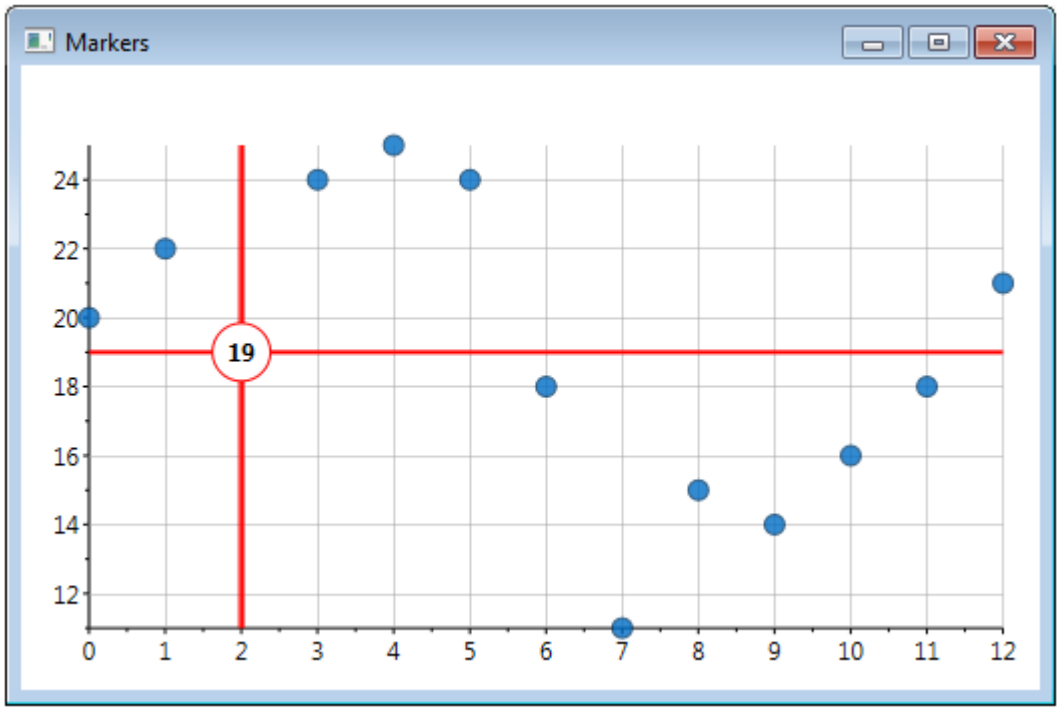
XAML

```
<!-- 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");
    }
}
```

鼠标和ChartPanel交互

ChartPanel具有鼠标交互的支持。ChartPanelAction（在线文档 'ChartPanelAction 枚举'）枚举指定图表面板对象可能的行为。ChartPanelAction 枚举包括以下成员：

通过Action（在线文档 'Action 类'）属性，我们可以生成一个可拖拽的元素，或者一个追随鼠标指针的元素。例如，在前面的示例中添加了Action，我们生成了最终用户可以移动的标记。

XAML

```
<!-- 垂直线 -->
   <c1chart:ChartPanelObject DataPoint="0,NaN" VerticalAlignment="Stretch"
       Action="LeftMouseButtonDrag" >
     <Border BorderBrush="Red" BorderThickness="3,0,0,0"
Margin="-1.5,0,0,0" />
   </c1chart:ChartPanelObject>
```

使用数据绑定很容易添加标签，显示当前的坐标：

XAML

```
  <!-- vertical line with coordinate label -->
   <c1chart:ChartPanelObject x:Name="xmarker" DataPoint="0,NaN"
VerticalAlignment="Stretch"
       Action="LeftMouseButtonDrag">
     <Border BorderBrush="Red" BorderThickness="3,0,0,0"
Margin="-1.5,0,0,0" >
       <TextBlock
         Text="{Binding RelativeSource={RelativeSource Self},
         Path=Parent.Parent.DataPoint.X,StringFormat='x=0.0;x=-0.0'}" />
     </Border>
   </c1chart:ChartPanelObject>
```

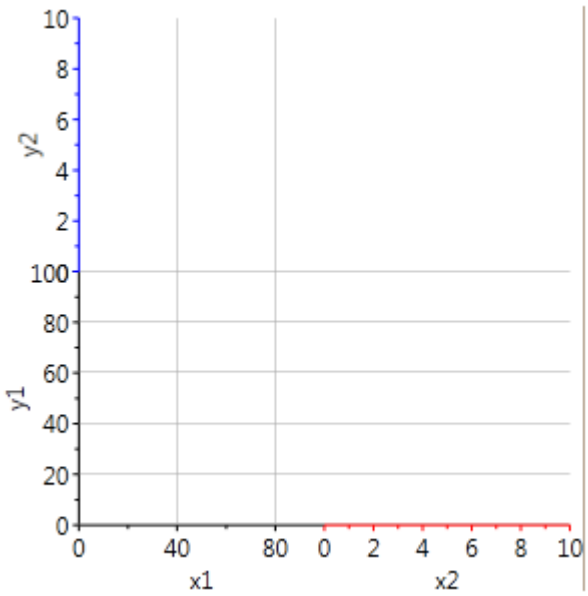
Attach属性允许关联元素可能的位置至最近的数据点。它可以关联到单一的坐标（X或Y）或同时关联到两个坐标坐标（X和Y）。

多绘图区

数据被绘制在图表的绘图区中。Plot区域是绘图区的一部分，由坐标轴限定并包含全部的绘图区元素（条形，柱形，折线等等。）。以前，图表只能有一个绘图区域，但现在有可能在同一个图表中有几个绘图区。

通常情况下，绘图区自动地基于PlotAreaIndex （在线文档 'PlotAreaIndex 属性'）属性进行创建。默认情况下，该属性的值为 0，不会为额外的坐标轴创建新的绘图区。仅仅是添加了坐标轴，例如，在主Y轴的左侧或者主X轴的底部。但如果您设置了PlotAreaIndex = 1，则新的坐标轴作为主轴添加在同一条线上。对于x-轴，辅助轴将位于右侧，对于y-轴，辅助轴位于顶部。

下面的例子说明了在同一直线上添加新的轴作为主轴：



XAML

```
<c1chart:C1Chart x:Name="chart" >
       <c1chart:C1Chart.View>
         <c1chart:ChartView>
           <!-- 主坐标轴  -->
           <c1chart:ChartView.AxisX>
             <c1chart:Axis Min="0" Max="100" Title="x1" />
           </c1chart:ChartView.AxisX>
           <c1chart:ChartView.AxisY>
             <c1chart:Axis Min="0" Max="100" Title="y1" />
           </c1chart:ChartView.AxisY>

            <!-- 辅助轴位于主X-轴的右侧 -->
           <c1chart:Axis x:Name="x2" Title="x2" PlotAreaIndex="1"
              AxisType="X" Min="0" Max="10" />

           <!-- 辅助轴位于主Y-轴的上方 -->
           <c1chart:Axis x:Name="y2" Title="y2" PlotAreaIndex="1"
              AxisType="Y" Min="0" Max="10" />

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

为添加数据，您需要指定坐标轴的名称（DataSeries.AxisX/AxisY）,之后数据将沿着辅助轴进行绘制。

绘图区尺寸

PlotArea的尺寸可以通过PlotAreaCollection类的ColumnDefinitions以及RowDefinitions集合指定。这一过程和操作标准的Grid控件类似。第一个集合包含列属性（宽度）。第二个集合包含行（高度）。默认情况下，绘图区域具有相同的宽度和相同的高度。

下面的示例演示如何以编程方式指定绘图区域的大小：

C#

```
// 宽度
    // 第一个绘图区域的宽度为默认值（填充可用空间）

    chart.View.PlotAreas.ColumnDefinitions.Add(new PlotAreaColumnDefinition());
      // 第二个绘图区的宽度为常量，100像素。
    chart.View.PlotAreas.ColumnDefinitions.Add(new PlotAreaColumnDefinition()
      { Width= new GridLength(100) });
    // 高度
    // 第一个绘图区的高度是1 *

    chart.View.PlotAreas.RowDefinitions.Add(new PlotAreaRowDefinition()
      { Height = new GridLength(1, GridUnitType.Star) });
    // 第二个绘图区的高度是 2*
    chart.View.PlotAreas.RowDefinitions.Add(new PlotAreaRowDefinition()
      { Height = new GridLength(2, GridUnitType.Star) });
```

绘图区外观

您可以通过Background属性修改PlotArea的外观，同时可以使用Stroke/StrokeThickness属性修改绘图区的边框。绘图区通过行/列进行引用（和Grid中访问元素相同）。

下面的示例演示如何修改绘图区外观：

XAML

```
<c1chart:ChartView.PlotAreas>
      <!-- row=0 col=0 -->
      <c1chart:PlotArea Background="#10FF0000" Stroke="Red" />
      <!-- row=1 col=0 -->
      <c1chart:PlotArea Row="1" Background="#1000FF00" />
      <!-- row=0 col=1 -->
      <c1chart:PlotArea Column="1" Background="#100000FF" />
      <!-- row=1 col=1 -->
      <c1chart:PlotArea Row="1" Column="1" Background="#10FFFF00"
Stroke="Yellow" />
    </c1chart:ChartView.PlotAreas>
```

性能优化

启用图表优化

C1Chart可以处理大批量的数据，但有时会导致性能问题。这些问题可以用SetOptimizationRadius()方法简单解决。使用这种方法可以减少重复数据点的数量。

下面的代码演示了如何使用该方法：

C#

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, 5);
```

渲染模式

C1Chart提供三种渲染模式，这样您可以控制您的图表的性能。有一个默认的渲染模式，支持所有图表类型，和两种高性能的渲染模式。高性能的渲染模式会让你更快地渲染你的图表，但它们也有一定的局限性。

渲染模式	局限性
Default	默认渲染模式。所有图表类型都支持。
Fast	位图渲染模式。高性能渲染模式。在当前版本只有折线图和符号图表支持。数据点标签，工具提示以及PlotElementLoaded事件将不可用。
Bitmap	位图模式。高性能渲染模式。此时此刻仅支持折线图和符号图。数据点标签，

	工具提示以及PlotElementLoaded事件将不可用。

执行批处理更新

你可以执行批处理更新，无需在图表的每一个变化发生时立即刷新，仅需要将您的代码放置在BeginUpdate()/EndUpdate() 方法之间，如下所示：

Visual Basic

```
C1Chart1.BeginUpdate()
    ' 改变或者格式化图表，添加数据等等。
    ...
    C1Chart1.EndUpdate()
```

C#

```
c1Chart1.BeginUpdate();
  //改变或者格式化图表，添加数据等等。
    ...
    c1Chart1.EndUpdate();
```

在设置了图表优化之后重新禁用优化

若要禁用已经设置了的图表优化如下：

Visual Basic

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, 2.0)
```

C#

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, 2.0);
```

您可以设置其为默认值，NaN，如下所示：

Visual Basic

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, double.NaN)
```

C#

```
LineAreaOptions.SetOptimizationRadius(c1Chart1, double.NaN);
```

绘图功能

C1Chart 具有内置的绘图功能引擎。为了使用内置的绘图功能引擎，必须在您的工程中添加对 C1.WPF.C1Chart.Extended.dll 或者 C1.Silverlight.Chart.Extended.dll的引用。

有各种不同类型的功能供不同的应用程序使用。C1Chart 提供创建许多应用程序的各种不同类型的功能。

有两种支持的功能类型：

1. 单变量显式函数
   单变量的显函数书写为写为y = f（x）的形式（参见YFunctionSeries 类）。
   一些例子包括：有理，线性，多项式，二次，对数，指数函数。
   通常由科学家和工程师使用，这些功能可用于许多不同类型的财务，预测，性能测量应用，等等。
2. 参数函数
   该函数由一组等式组成，比如说y = f1(t) 以及 x = f2(t)，t是函数f1和f2的变量/坐标。
   参数函数是特殊类型的函数，因为在一个单独的变量的单独作用下，该函数的函数是特殊类型的。
   它们被用来代表各种情况，在数学和工程，从传热，流体力学，电磁理论，行星运动和相对论方面，仅举几例。

关于参数函数的更多信息（参见ParametricFunctionSeries 类）。

使用一个代码字符串来定义一个函数

当一个解释性的代码字符串是用来定义一个函数类的功能，字符串将编译生成的代码并动态地包含到应用程序。执行速度将与其他编译码相同。

简单而言，对于单变量的显函数，将使用YFunctionSeries类的对象。该对象有一个编码属性，FunctionCode。对于 YFunction 对象，自变量总是假定为"x"。

对于参量函数，双方程必须使用ParametricFunctionSeries 类对象的定义。该对象具有两个属性，每一个用于一个坐标。两个属性，XFunctionCode 以及YFunctionCode 接受的代码中，自变量总是假定为“t”。

计算函数的值

您可以通过CalculateValue()方法计算参量函数以及YFunction方程的值。

下面的代码演示了CalculateValue()方法：

C#

```
class MySeries : FunctionSeries
{
    void SomeMethod()
    {
       CalculateValue(...);
    }
}
```

保存和导出C1Chart

将图表导出为PDF格式

为导出图表为位图图像，并使用C1Pdf库将该图像导出为PDF，请使用以下代码：

C#

```
// 保存图表图像至文件流
    MemoryStream ms = new MemoryStream();
    chart.SaveImage(ms, ImageFormat.Png);

    // 从文件流创建图像实例
    var img = System.Drawing.Image.FromStream(ms);

    // 创建并保存PDF文档
    C1PdfDocument pdf = new C1PdfDocument();
    pdf.DrawImage( img, new System.Drawing.RectangleF(0,0,img.Width,img.Height));
    pdf.Save("doc.pdf");
```

导出图表图像

您可以通过RenderTargetBitmap 方法导出图表的图像，如以下代码所示：

Visual Basic

```
Dim bm As New RenderTargetBitmap(CInt(c1Chart1.ActualWidth),
CInt(c1Chart1.ActualHeight), 96, 96, PixelFormats.[Default])
bm.Render(c1Chart1)

Dim enc As New PngBitmapEncoder()
enc.Frames.Add(BitmapFrame.Create(bm))

Dim fs As New FileStream("chart.png", FileMode.Create)
enc.Save(fs)
```

C#

```
RenderTargetBitmap bm = new RenderTargetBitmap(
      (int)c1Chart1.ActualWidth,(int)c1Chart1.ActualHeight,
        96, 96, PixelFormats.Default);
bm.Render(c1Chart1);

PngBitmapEncoder enc = new PngBitmapEncoder();
enc.Frames.Add(BitmapFrame.Create(bm));

FileStream fs = new FileStream("chart.png", FileMode.Create);
enc.Save(fs);
```

将C1Chart保存为 .png 文件

为将C1Chart保存为 .png 文件，使用下面的代码：

Visual Basic

```
' 保存图像至文件实例
Using stm = System.IO.File.Create("chart.png")
    c1Chart1.SaveImage(stm, C1.WPF.C1Chart.Extended.ImageFormat.Png)
End Using
```

C#

```
// 保存图像至文件实例
      using (var stm = System.IO.File.Create("chart.png"))
      {
        c1Chart1.SaveImage(stm, C1.WPF.C1Chart.Extended.ImageFormat.Png);
      }
```

生成系列

注意：本主题的示例可以在博文“图表自动系列生成（MVVM）”中找到。

对于使用MVVM的开发者，系列可以完全在ViewModel中通过两个ChartData对象属性生成：SeriesItemSource 以及SeriesItemTemplate。

在一个实际的场景中，您希望为每一年绘制一个不同的数据系列，但是不同年份的数量在设计时并不确定，则您可以在ViewModel中确定年份的数量。

首先我们要着重介绍一些在绑定属性至ViewModel的元素。在突出显示的XAML标记和代码之后，是包含全部必须的标记和代码的MVVM自动生成系列的主题。

在以下的XAML标记中，您可以但看到这两个属性：

XAML

```
<c1:C1Chart Name="c1Chart1">
    <c1:C1Chart.Data>
        <c1:ChartData SeriesItemsSource="{Binding SeriesDataCollection}">
         <c1:ChartData.SeriesItemTemplate>
                <DataTemplate>
                    <c1:DataSeries Label="{Binding Year}" ValuesSource="{Binding
Values}" />
              </DataTemplate>
    </c1:ChartData.SeriesItemTemplate>
        </c1:ChartData>
    </c1:C1Chart.Data>
```

```
    <c1:C1ChartLegend DockPanel.Dock="Right" />
</c1:C1Chart>
```

SeriesItemsSource 以及 SeriesItemTemplate 属性均在ChartData对象上设置。

SeriesItemTemplate. SeriesItemsSource绑定到ViewModel，同时包括SeriesItemTemplate的Label以及ValuesSource 属性。

重点介绍ViewModel的两个章节，第一个创建SeriesData的ObservableCollection：

C#

```
public ObservableCollection<SeriesData> SeriesDataCollection
        {
            get
            {
                if (_seriesDataCollection == null)
                {
                    _seriesDataCollection = new ObservableCollection<SeriesData>();
                    for (int i = 0; i < ViewModelData.NUM_SERIES; i++)
                        _seriesDataCollection.Add(new SeriesData(2010 + i));
                }
                return _seriesDataCollection;
            }
        }
```

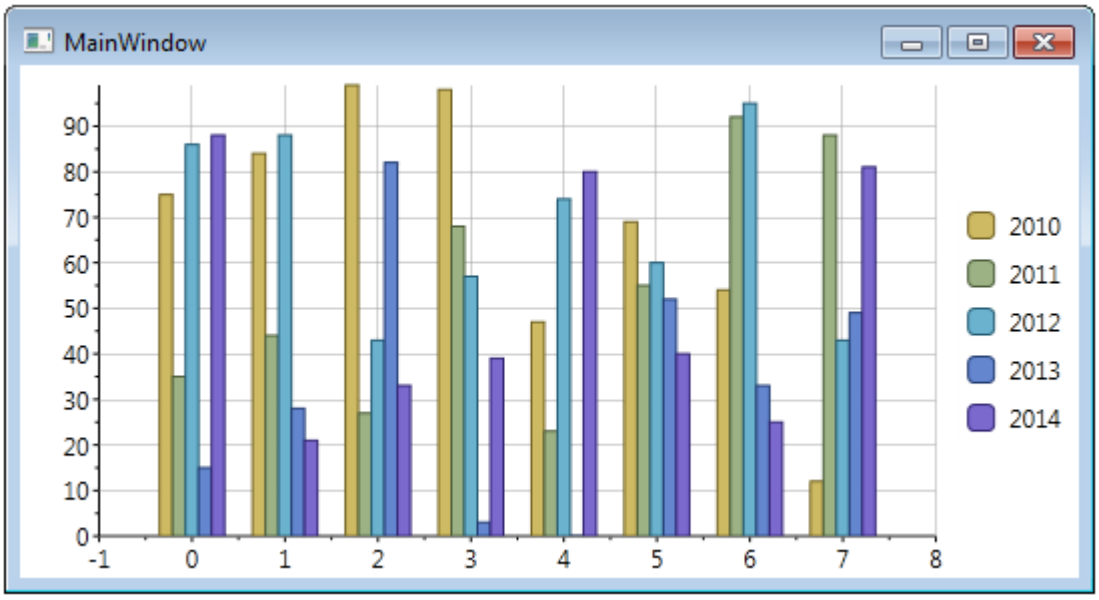
第二段代码是我们的自定义业务对象。它包含年份和数据值：

C#

```
public class SeriesData : INotifyPropertyChanged
    {
        int _year;
        double[] _values;

        public SeriesData(int year)
        {
            _year = year;

            _values = new double[ViewModelData.NUM_POINTS];
            for (int i = 0; i < ViewModelData.NUM_POINTS; i++)
                _values[i] = ViewModelData.Rnd.Next(0, 100);
        }

        public int Year
        {
            get { return _year; }
            set
            {
                if (_year != value)
                {
                    _year = value;
                    OnPropertyChanged("Year");
                }
```

```
        }
    }

    public double[] Values
    {
        get { return _values; }
        set
        {
            if (_values != value)
            {
                _values = value;
                OnPropertyChanged("Values");
            }
        }
    }
```

当您运行您的程序，或运行该示例，它应该像以下图像所示：



通过MVVM自动生成序列

本主题假设您已经创建了一个新的Visual Studio工程，并为您的工程添加了适当的引用。

步骤1）创建标记

这是让您开工的XAML标记：

```
XAML
```

```
<c1:C1Chart Name="c1Chart1">
    <c1:C1Chart.Data>
        <c1:ChartData SeriesItemsSource="{Binding SeriesDataCollection}">
         <c1:ChartData.SeriesItemTemplate>
                <DataTemplate>
                    <c1:DataSeries Label="{Binding Year}" ValuesSource="{Binding
Values}" />
               </DataTemplate>
    </c1:ChartData.SeriesItemTemplate>
        </c1:ChartData>
    </c1:C1Chart.Data>
    <c1:C1ChartLegend DockPanel.Dock="Right" />
</c1:C1Chart>
```

请注意，SeriesItemsSource以及SeriesItemTemplate属性在ChartData 对象上进行设置，他们将绑定到具体的值。

步骤2）创建视图模型

接下来，您需要创建您的项目视图。

右键单击您的项目名称，选择添加新项|选择代码文件，将其命名为ViewModel.cs，并单击“确定”。

将下面的代码添加到您的代码文件中，以创建视图模型：

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;
using System.Collections.ObjectModel;

namespace ChartAutomaticSeries
{
    public static class ViewModelData
    {
        public static int NUM_SERIES = 5;
        public static int NUM_POINTS = 8;

        public static Random Rnd = new Random();

        private static ChartModelData _data;

        public static ChartModelData ChartData
        {
            get
            {
                if (_data == null)
                {
                    _data = new ChartModelData();
                }
                return _data;
            }
        }
```

```
}

public class ChartModelData
{
    public ObservableCollection<SeriesData> SeriesDataCollection
    {
        get
        {
            if (_seriesDataCollection == null)
            {
                _seriesDataCollection = new ObservableCollection<SeriesData>();
                for (int i = 0; i < ViewModelData.NUM_SERIES; i++)
                    _seriesDataCollection.Add(new SeriesData(2010 + i));
            }
            return _seriesDataCollection;
        }
    }
    private ObservableCollection<SeriesData> _seriesDataCollection;
}

public class SeriesData : INotifyPropertyChanged
{
    int _year;
    double[] _values;

    public SeriesData(int year)
    {
        _year = year;

        _values = new double[ViewModelData.NUM_POINTS];
        for (int i = 0; i < ViewModelData.NUM_POINTS; i++)
            _values[i] = ViewModelData.Rnd.Next(0, 100);
    }

    public int Year
    {
        get { return _year; }
        set
        {
            if (_year != value)
            {
                _year = value;
                OnPropertyChanged("Year");
            }
        }
    }

    public double[] Values
    {
        get { return _values; }
        set
```

```
            {
                if (_values != value)
                {
                    _values = value;
                    OnPropertyChanged("Values");
                }
            }
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

        private void OnPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        #endregion
    }
}
```

步骤3）添加代码

切换回您的MainWindow.xaml文件。右键单击“页面”，并从“上下文”菜单中选择“查看代码”。
编辑现有的代码，以便它类似于以下：

C#

```
public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            this.DataContext = new ChartModelData();
        }
    }
}
```

AutoGenerateSeries 属性

属性AutoGenerateSeries（'autogenerateseries财产在在线文档）指定是否自动创建的系列。默认情况下，AutoGenerateSeries属性的值空，仅在Children集合为空时生成数据系列。当即将生成这些数据系列时，图表将分析Data.ItemsSource（或者C1Chart.DataContext）集合，并为每一个支持的属性类型（数值，日期时间）创建数据系列。

为了控制系列生成的过程，可以使用Binding属性指定哪一个属性应当被绘制在图表上。关于Binding属性的更多信息，请参见数据系列绑定。

样式与外观

当图表的数据和坐标轴轴被正确格式化时，它的元素可以被定制，使其看起来更清晰，更专业。以下主题包括自定义图表外观的任务。

图表资源Key

内置主题和资源有几个注册的资源Key。这些Key包括画刷，边框，以及其他元素，可以进行定制以表示一个独特的外观。当定制主题时，如果没有显示地指定资源Key将返回到使用默认值。包含的资源关键字及其描述将在在以下主题中说明。
下表将描述图表控件的图表资源Key及其元素，如图表区、绘图区、坐标轴和图例区。

图表资源Key

资源Key	描述
C1Chart_Foreground_Color	表示C1Chart的前景颜色。
C1Chart_Background_Color	表示C1Chart的前景颜色。
C1Chart_Background_Brush	表示C1Chart的背景画刷。
C1Chart_Foreground_Brush	表示C1Chart的前景画刷。
C1Chart_Border_Brush	表示C1Chart的边框画刷。
C1Chart_Border_Thickness	表示C1Chart的边框宽度（全部四边）。
C1Chart_CornerRadius	表示图表的圆角半径（全部四个角）。
C1Chart_Padding	表示C1Chart的内边距。
C1Chart_Margin	表示C1Chart的边距。

图例资源Key

资源Key	描述
C1Chart_LegendBackground_Brush	表示 C1Chart 图例区的背景画刷。
C1Chart_LegendForeground_Brush	表示C1Chart图例区的前景画刷。
C1Chart_LegendBorder_Brush	表示C1Chart图例区的边框画刷。
C1Chart_LegendBorder_Thickness	表示C1Chart图例区的边框宽度（全部四边）。
C1Chart_Legend_CornerRadius	表示C1Chart图例区的边框圆角半径（全部四个角）。

图表区域资源Key

资源Key	描述
C1Chart_ChartAreaBackground_Brush	表示ChartArea的背景画刷。
C1Chart_ChartAreaForeground_Brush	表示当鼠标悬停经过时，ChartArea的前景画刷。
C1Chart_ChartAreaBorder_Brush	表示ChartArea的边框画刷。
C1Chart_ChartAreaBorder_Thickness	表示ChartArea的边框宽度。
C1Chart_ChartArea_CornerRadius	表示ChartArea的圆角半径（全部四个角）。

C1Chart_ChartArea_Padding	表示ChartArea的内边距。

绘图区资源Key

资源Key	描述
C1Chart_PlotAreaBackground_Brush	表示 PlotArea的背景画刷。

绘图区元素自定义调色板Key

资源Key	描述
C1Chart_CustomPalette	表示绘图区元素的自定义调色板。

Axis Keys

资源Key	描述
C1Chart_AxisMajorGridStroke_Brush	表示 AxisMajorGridStroke的画刷。
C1Chart_AxisMinorGridStroke_Brush	表示AxisMinorGridStroke 的画刷。

图表样式

绘图区元素支持WPF Style，这是控制图表外观的方便方式。

MouseOver 样式

以下示例演示如何创建一个样式，该样式设置一个PlotElement的Stroke属性的值为Black：

XAML

```
<Window.Resources>
   ...
   <Style x:Key="mouseOver" TargetType="{x:Type c1c:PlotElement}">
          <!-- 默认黑色轮廓 -->
          <Setter Property="Stroke" Value="Black" />
          <Style.Triggers>
          <!-- 当鼠标悬停在元素上访时，使其显示粗的红色外框    -->
                 <Trigger Property="IsMouseOver" Value="true">
                        <Setter Property="Stroke" Value="Red" />
                        <Setter Property="StrokeThickness" Value="3" />
                        <Setter Property="Canvas.ZIndex" Value="1" />”
                 </Trigger>
          </Style.Triggers>
   </Style>
</Window.Resources>
```

注意：当设置您为PlotElement类型的样式的TargetType属性时，如果不给您的样式分配一个x:Key，则该样式将应用到全部的PlotElement元素。

为了应用鼠标悬停样式至数据系列，您可以使用SymbolStyle（在线文档 'SymbolStyle 属性'）属性，如下所示：

XAML

```
<c1c:DataSeries … SymbolStyle="{StaticResource mouseOver}"/>
```

图表主题

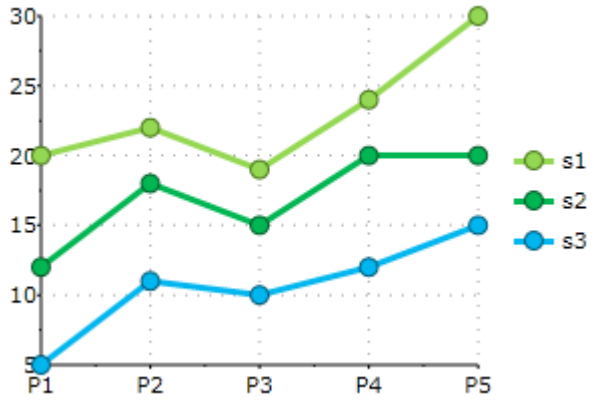
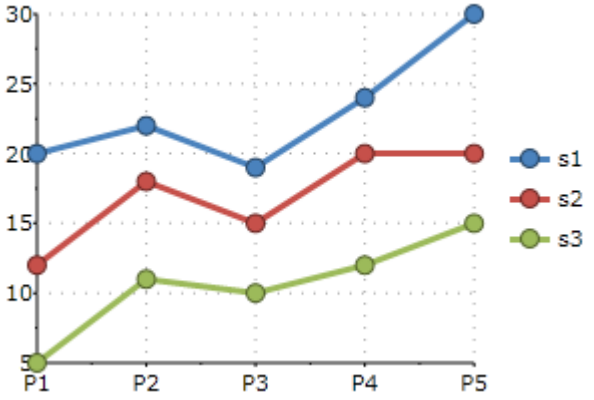
WPF及Silverlight版Chart集成了多种主题，包括Office 2007，BureauBlack，以及Office 2013的主题，允许您自定义图表的外观。内置主题在WPF文档入门 章节描述并展示。

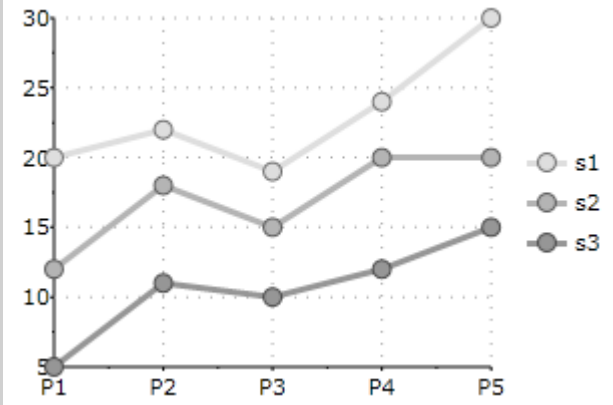
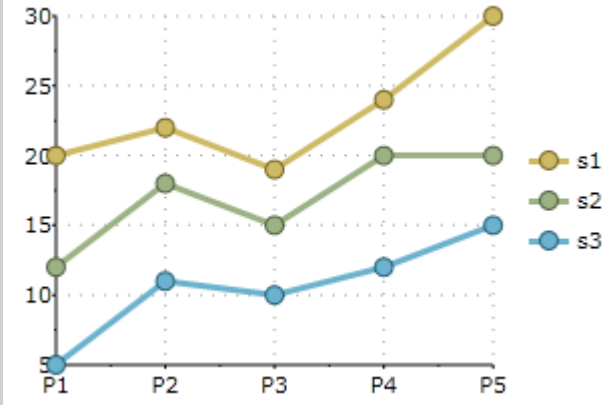
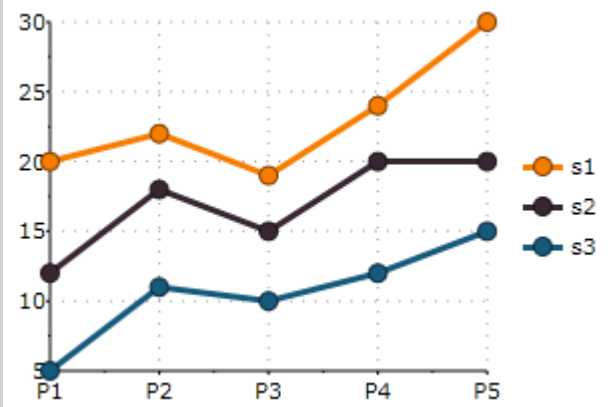
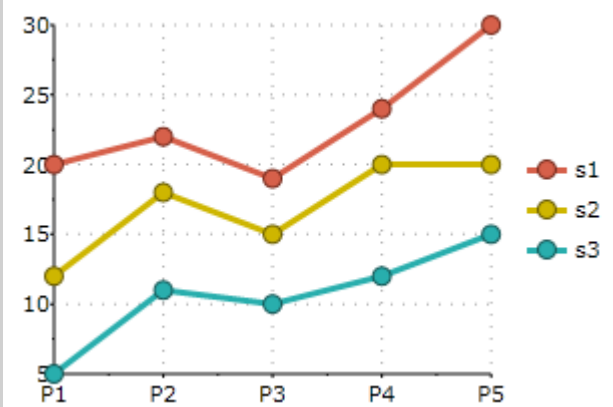
ComponentOne主题使您可以将主题应用于仅一个控件，或是整个应用程序。

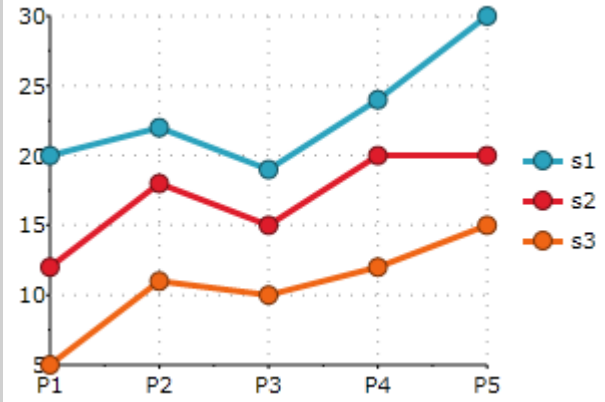
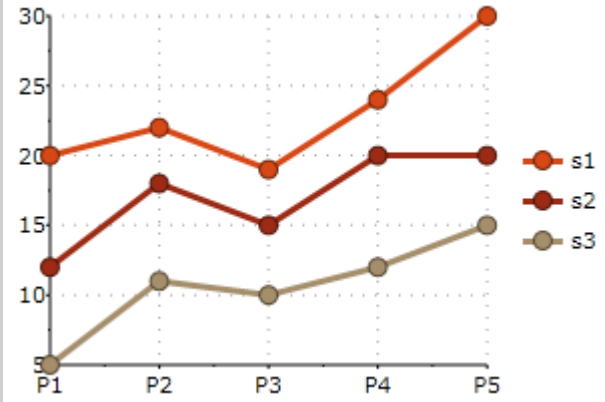
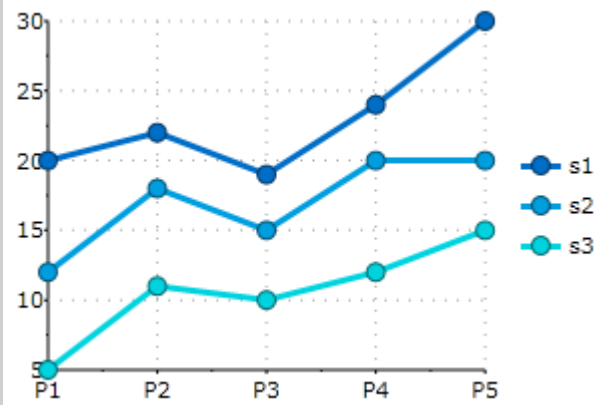
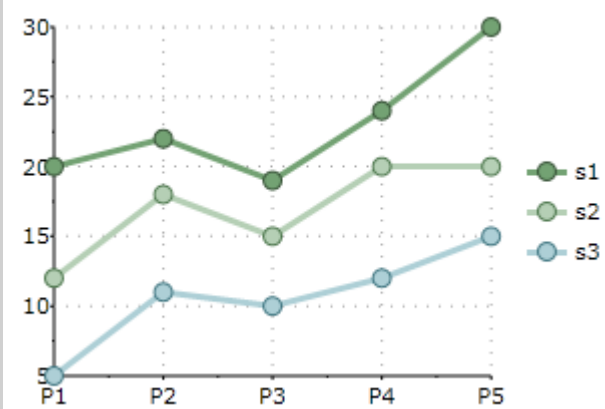
数据序列颜色生成

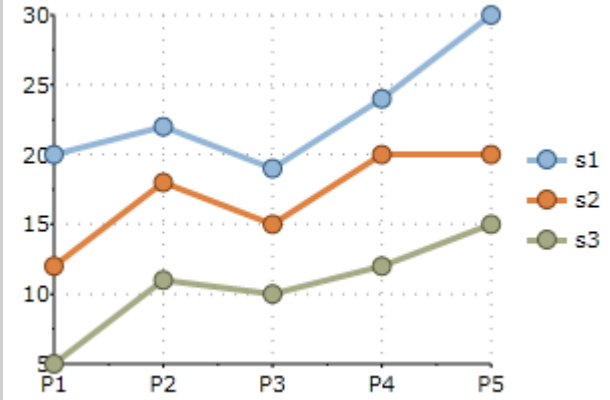
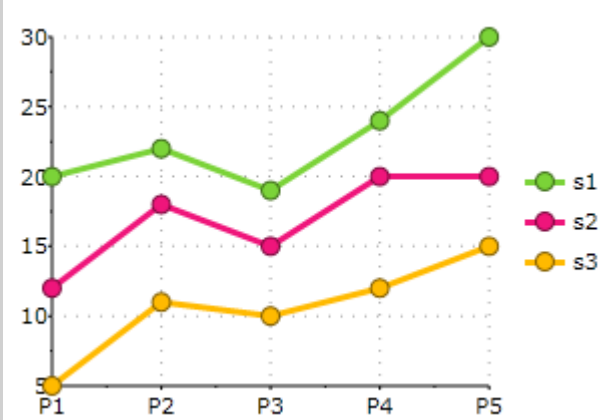
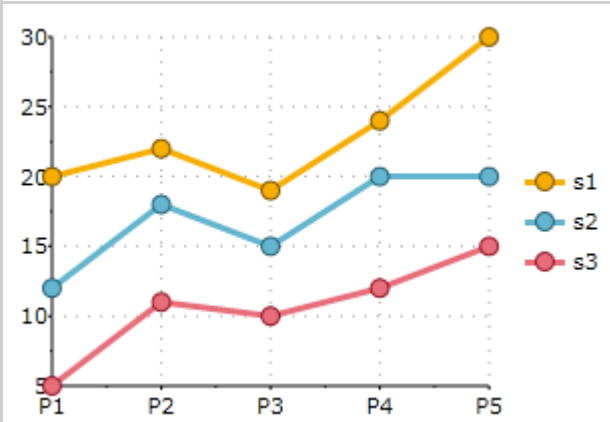
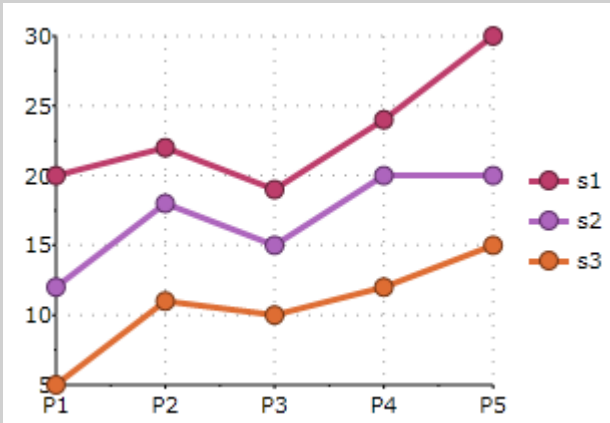
数据系列的配色方案可以通过Palette（在线文档 'Palette 属性'）属性进行选择。默认情况下，C1Chart采用ColorGeneration.Default 设置。剩下的选项模仿微软Office的颜色主题。

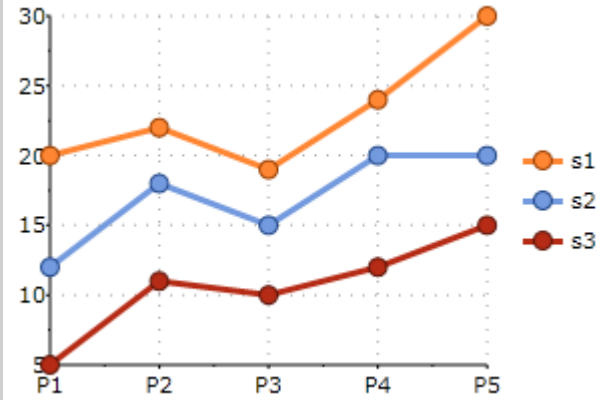
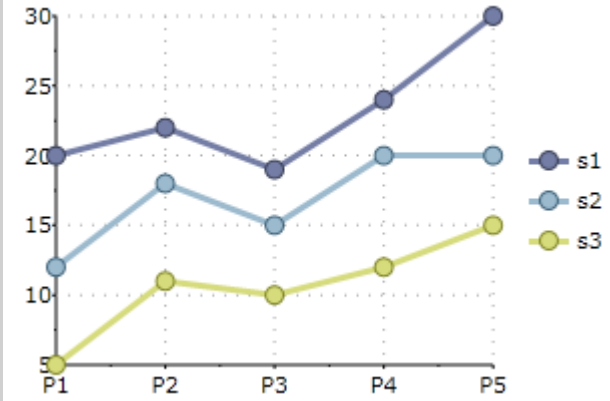
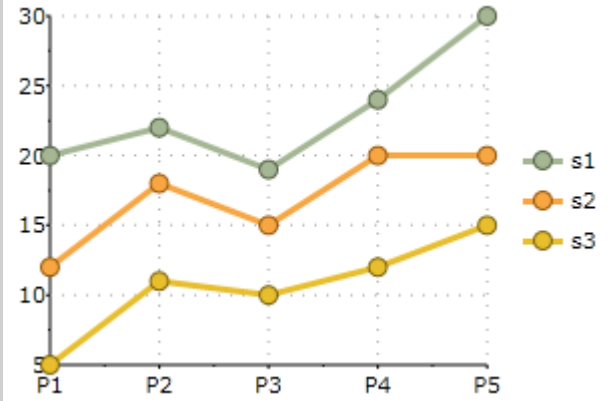
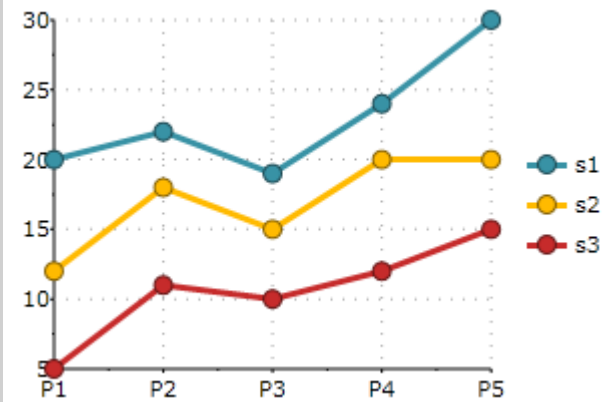
下面列出数据系列的可用颜色方案：

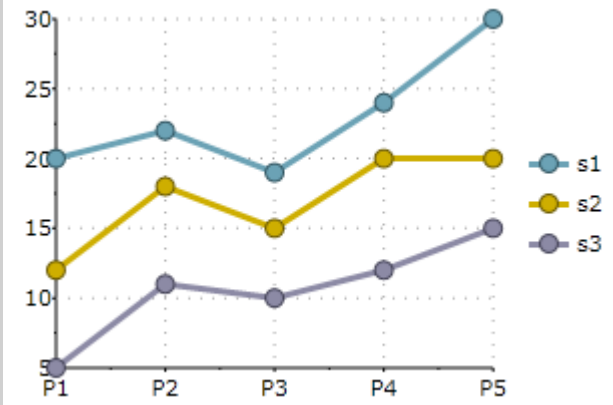
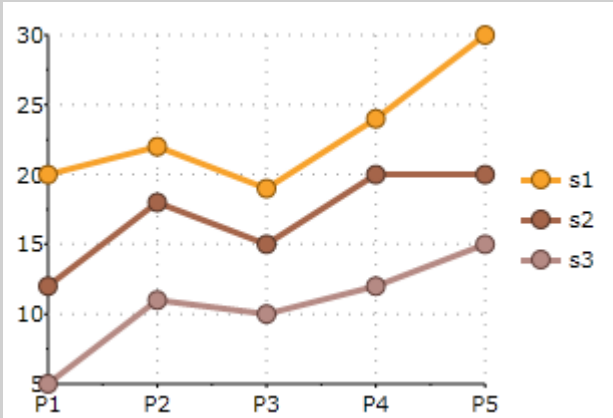
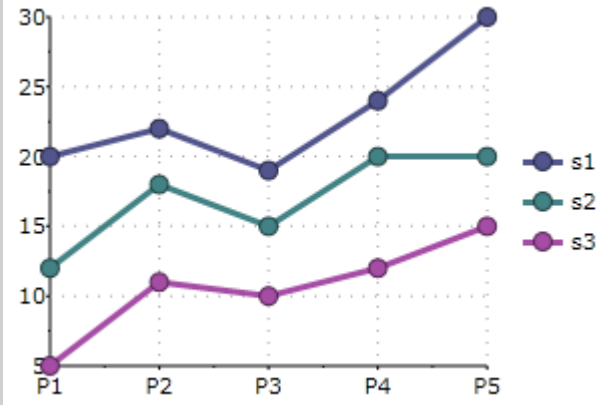
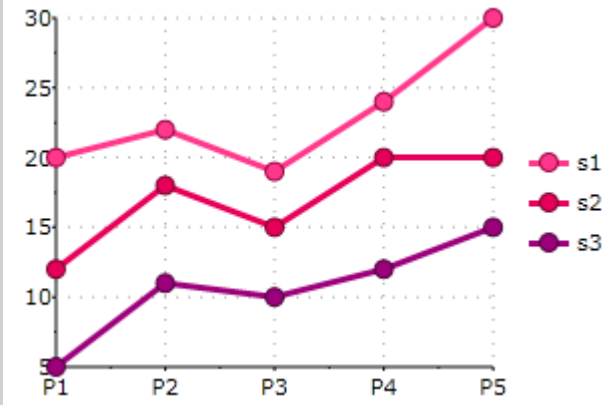
颜色生成设置	描述或预览
Default	当C1Chart.ColorGeneration设置为“Default”，图表在设置了主题时使用主题调色板，否则将应用“Apex”调色板。
Standard	
Office	

GrayScale	
Apex	
Aspect	
Civic	

Concourse	
Equity	
Flow	
Foundry	

Median	
Metro	
Module	
Opulent	

Oriel	
Origin	
Paper	
Solstice	

Technic	
Trek	
Urban	
Verve	

改变绘图元素颜色

为了改变分配给绘图区元素，比如说条形和饼图切片（取决于图表类型），您既可以改变Palette（在线文档 'Palette 属性'）属性为某一个预定义的调色板值，也可以创建您自己的自定义调色板，比如说：

C#

```
Brush[] customBrushes = new Brush[2] { Brushes.Blue, Brushes.Orange };
c1Chart1.CustomPalette = customBrushes;
```

格式化图表

前一章节介绍了Theme，您可以容易地快速容易地选择图表的外观。Theme（在线文档 'Theme 属性'）以及Palette（在线文档 'Palette 属性'）属性提供了一个内置选项的长长的列表，这些内置的选项被精心的研发以便开发人员可以通过最小的精力即可获得非常棒的结果。

在大多数应用程序中，您将选择设置为最接近于您的应用程序的主题和调色板属性的组合，然后在必要时自定义一些项目。您可以自定义的项目包括：

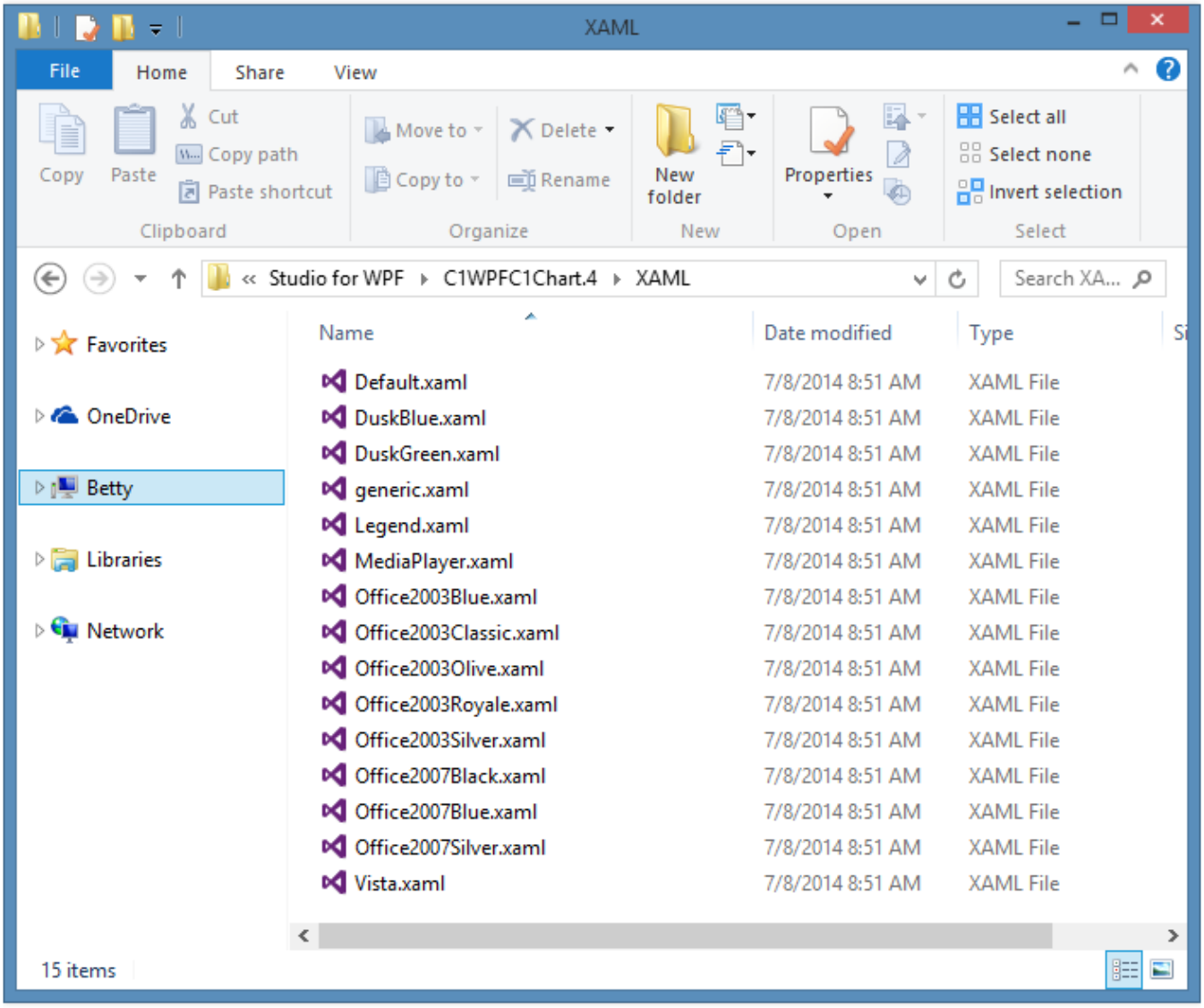
1. 坐标轴标题：坐标轴标题是UIElement对象。您可以直接定制，并具有完全的灵活性。在Simple Charts，时间系列图表以及散点图主题中的图表示例使用到了TextElement对象，但实际上您可以使用到许多其他元素，包括面板容器类型比如说Border以及Grid对象。有关坐标轴标题的更多信息，参见坐标轴标题主题。
2. 坐标轴：在Simple Charts，时间系列图表以及散点图主题中的图表示例演示了如何自定义坐标轴范围，标注旋转角度以及标注格式。所有这些都可以通过暴露在AxisX 以及 AxisY属性上的Axis对象访问。关于C1Chart的坐标轴的更多信息，请参见坐标轴。
   C1Chart控件具有通常的字体属性，用于确定标注如何沿着坐标轴显示（FontFamily，FontSize，等等）。如果您需要对标注外观的更多控制，Axis对象也暴露了一个AnnoTemplate属性，该属性可以被用来进一步自定义标注。
3. 网格线：网格线由Axis属性控制。针对于主网格线和次要网格线，分别有属性进行控制（MajorGridStrokeThickness，MajorGridStrokeThickness，MinorGridStrokeThickness，MinorGridStrokeThickness，等等 ）。关于网格线的更多信息，见网格线。
4. 刻度线：刻度线也由Axis属性进行控制。针对于主刻度线和次要刻度线，分别有属性进行控制（MajorTickStroke，MajorTickThickness，MinorTickStroke，MinorTickThickness，等等）。有关刻度线的更多信息，参见刻度线。

XAML元素

有几个辅助的XAML元素随同WPF版Chart安装。这些元素包括模版和主题，位于WPF安装路径下，默认情况下，为C:\Program Files\ComponentOne\WPF Edition\C1WPFChart\XAML。例如，你可以将这些元素整合到你的项目中，比如说，基于Office2007主题创建您自己的主题。有关内置主题的更多信息，请参见图表主题。

包含辅助XAML元素

下列辅助XAML元素包含在WPF版Char中，位于C:\Program Files\ComponentOne\WPF Edition\C1WPFChart\XAML。



时间系列图表

时间系列图表沿x轴显示时间。这是一个很常见的图表类型，用于显示随时间变化的值的变化。
大多数时间序列图显示固定的时间间隔（每年，每月，每周，每日）。在这种情况下，时间序列图基本上等同于一个简单的值类型图，就像上面描述的一个简单的值类型图。唯一的区别是，而不是显示类别沿轴线，图表将显示日期或时间。（如果时间间隔不定，然后图表将成为一个XY图，这一点将在下一节中描述。）我们将通过创建一些时间系列图表以演示如何使用。

步骤1）选择图表类型：

代码清除任何现有的序列，然后设置图表类型：

C#

```
public Window1()
    {
        InitializeComponent();
        // 清除当前图表
        c1Chart.Reset(true);
```

```
        // 设置图表类型
        c1Chart.ChartType = ChartType.Column;
    }
```

步骤2）设置坐标轴：

我们将从获取两个坐标轴的引用开始，就像在之前的示例中那样。回想一下，条形图类型使用的是反转坐标轴（值显示在Y-轴上）：

C#

```
//获取坐标轴
Axis valueAxis = c1Chart.View.AxisY;
Axis labelAxis = c1Chart.View.AxisX;
     if (c1Chart.ChartType == ChartType.Bar)
     {
         valueAxis = _c1Chart.View.AxisX;
         labelAxis = _c1Chart.View.AxisY;
      }
```

下一步我们将指定标题至坐标轴。坐标轴标题是UIElement对象而不是简单的文本。我们将使用CreateTextBlock()方法建立坐标轴标题，就像我们之前做的那样。我们还将设置标注格式，最小值和主单位。唯一的区别是我们将用一个更大的间隔来标记值之间的刻度：

C#

```
// 配置标签轴
labelAxis.Title = CreateTextBlock("Date", 14, FontWeights.Bold);
labelAxis.AnnoFormat = "MMM-yy";

// 设置值坐标轴
valueAxis.Title = CreateTextBlock("Amount ($1000)", 14, FontWeights.Bold);
valueAxis.AnnoFormat = "#,##0 ";
valueAxis.MajorUnit = 1000;
valueAxis.AutoMin = false;
valueAxis.Min = 0;
```

步骤3）添加一个或多个数据序列

这一次，我们将使用之前定义的第二个Data-Provider方法：

C#

```
// 获取数据
var data = GetSalesPerMonthData();
```

接下来，我们要沿着标签轴显示日期。为此，我们将使用LINQ语句在我们的数据记录中检索不同的日期值。然后结果被转换为数组设置给该标签轴的ItemsSource属性。

C#

```
c1Chart.ChartData.ItemNames = (from r in data select r.Date.ToString("MMM-
yy")).Distinct().ToArray();
```

注意，我们使用了LINQ的Distinct运算符删除重复的日期值。这个操作时必须的，因为我们的数据对每一天的每一种产

品包含一条记录。

现在我们可以创建将被添加到图表的实际的DataSeries对象。每一个系列将显示一个给定的产品的收入。这也可以通过Linq语句完成，该Linq语句相对于以往的将略微复杂，但是提供了一个很好的范例，展示了Linq语法的强大：

C#

```
// 为每一种产品添加一个数据系列（收入）
 var products = (from p in data select p.Product).Distinct();
 foreach (string product in products)
 {
   var ds = new DataSeries();
   ds.Label = product;
   ds.ValuesSource = (
     from r in data
       where r.Product == product
       select r.Revenue).ToArray();
      c1Chart.ChartData.Children.Add(ds);
 }
```

代码通过在数据源中构建一个产品列表开始。接下来它为每一个产品创建了一个DataSeries。数据系列的标签简单地显示产品的名称。实际的数据是通过过滤的记录，属于当前的产品并获取他们的Revenue属性。和之前一样，结果设置给数据系列的ValuesSource属性。

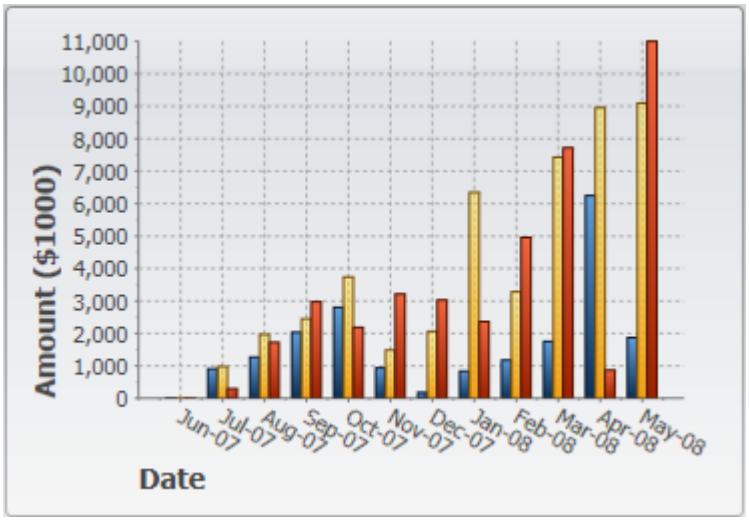
步骤4）调整图表的外观

再次，我们将通过设置主题和调色板属性来快速配置图表外观：

C#

```
c1Chart.Theme = c1Chart.TryFindResource(
            new ComponentResourceKey(typeof(C1.WPF.C1Chart.C1Chart),
            "Office2007Black")) as ResourceDictionary;}
```

这是最终的结束代码，以生成我们的时间系列图表。其结果应该是类似于下面的图像：



注意：AnnoAngle属性被设置为"30"，使图像中的Y轴上的标签有更多空间可以显示。

在每个月的第一天显示数据标签

为了仅在每个月的第一天上显示数据标签，请使用以下代码：

Visual Basic

```
c1Chart1.View.AxisX.IsTime = True
c1Chart1.View.AxisX.AnnoFormat = "MMM-dd"
' 当MajorUnit=31时时间轴图表的每月的总天数应该是变化的并且标记每月第一天
c1Chart1.View.AxisX.MajorUnit = 31
```

C#

```
c1Chart1.View.AxisX.IsTime = true;
c1Chart1.View.AxisX.AnnoFormat = "MMM-dd";
// 当MajorUnit=31时时间轴图表的每月的总天数应该是变化的并且标记每月第一天
c1Chart1.View.AxisX.MajorUnit = 31;
```

趋势线

注意：要使用趋势线的功能，就要添加到 C1.WPF.C1Chart.Extended.dll 或 C1.Silverlight.Chart.Extended.dll 的引用至您的工程。

C1Chart 支持十种不同的支持趋势线，使用该功能将非常容易。您所要做的就是控制如何添加它们至您的图表，以及他们的外观如何。

C1Chart控件支持两组不同的趋势线：回归和非回归。平均，最大，最小，和移动平均趋势线为非回归趋势线。

多项式，指数，对数，功率，和傅里叶函数为回归趋势线将分析函数趋势的近似值。

注意：趋势线不能用于所有类型的二维图表；通常，它们用在X-Y折线图，条形图，或散点图中。

添加趋势线至C1Chart

添加趋势线和添加一个数据系列非常相似。C1Chart将自动基于数值计算并绘制趋势。默认情况下，趋势线将有一个多项式FitType，其Order属性的值为2。下面的代码演示如何添加趋势线：

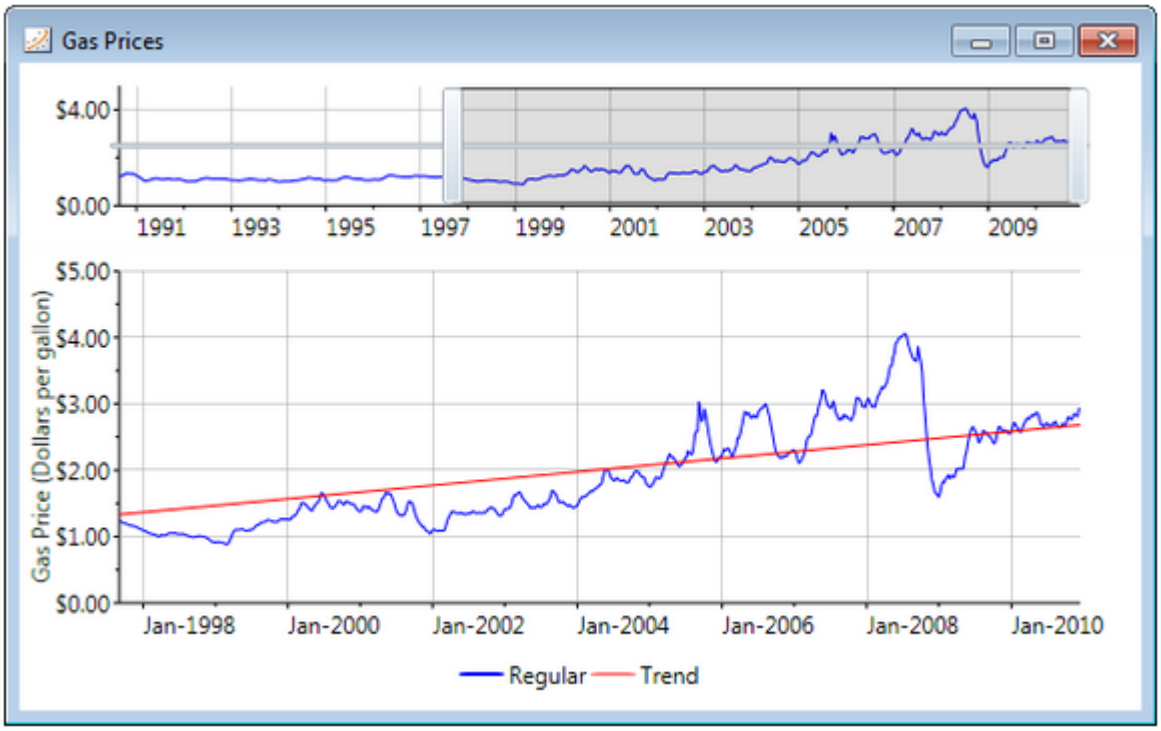
Visual Basic

```
'add trend line
Dim tl As New TrendLine()
tl.Label = "Trendline"
tl.ConnectionStroke = New SolidColorBrush(Colors.Red)
tl.XValuesSource = myXValues
tl.ValuesSource = myValues
chart.Data.Children.Add(tl)
```

C#

```
//添加趋势线
TrendLine tl = new TrendLine();
tl.Label = "Trendline";
tl.ConnectionStroke = new  SolidColorBrush(Colors.Red);
tl.XValuesSource = myXValues;
tl.ValuesSource = myValues;
chart.Data.Children.Add(tl);
```

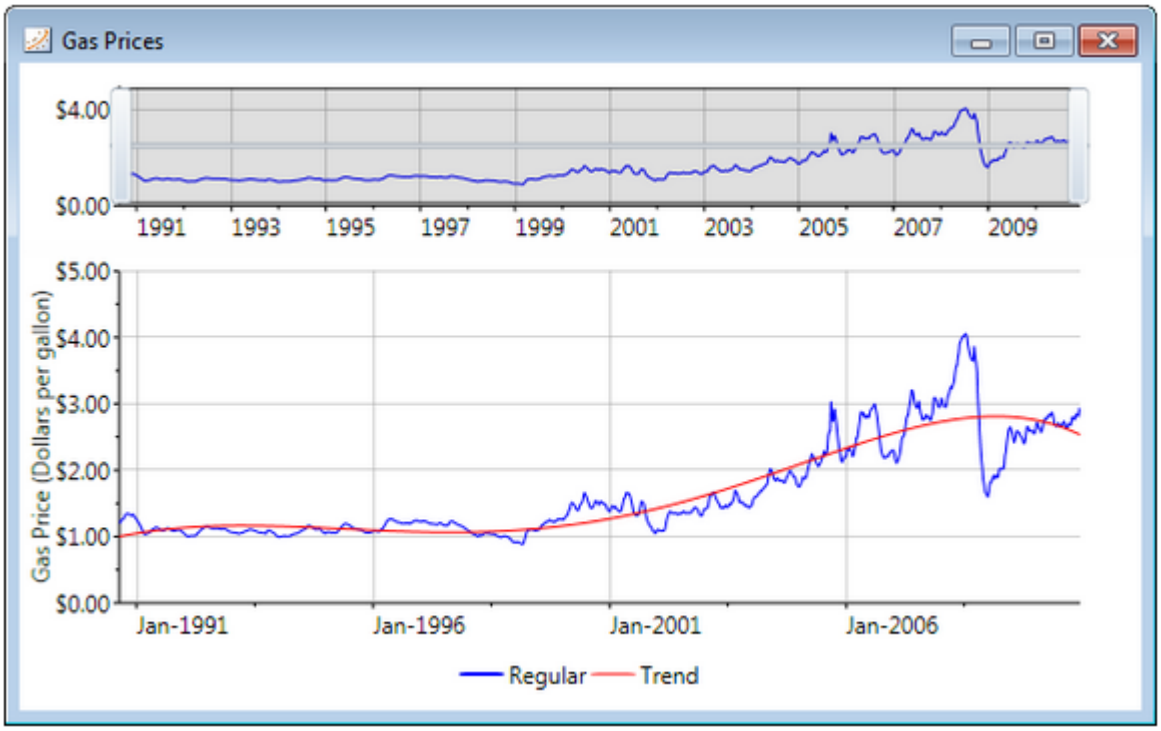
你可以看到下面的图像，默认的趋势线没有紧密地靠近数据：



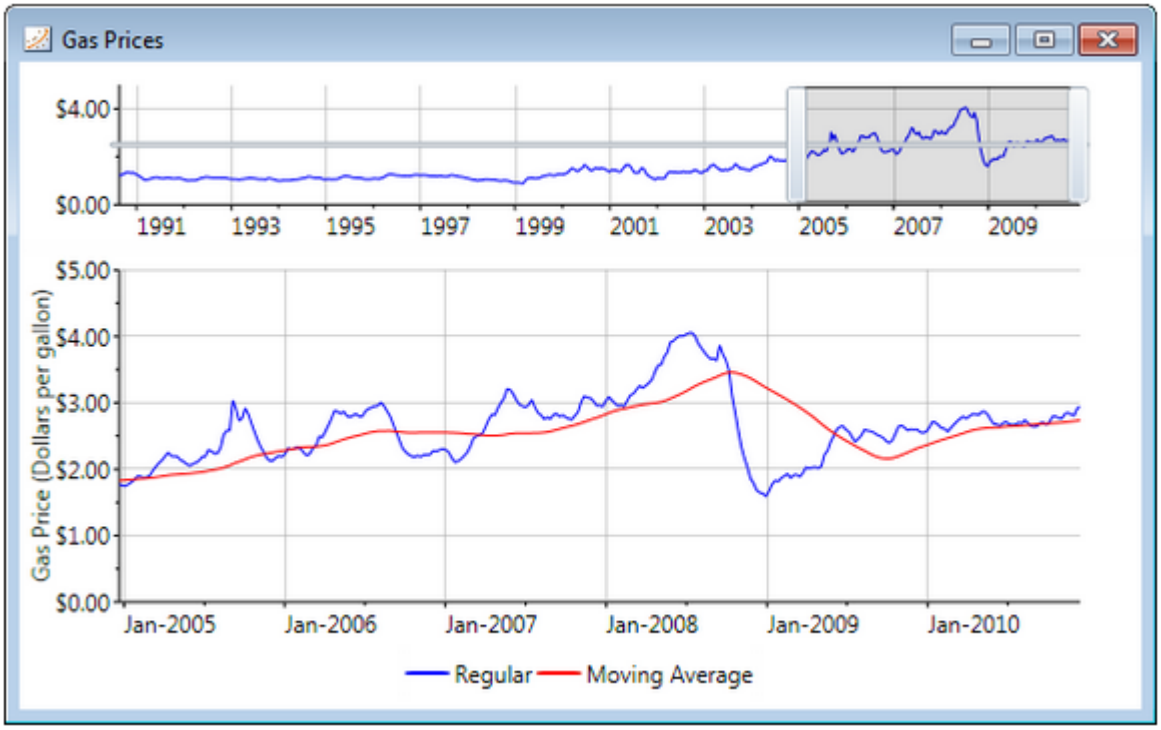
你可以通过改变Order属性来获得更好的拟合。您可以指定FitType以及Order属性以获取您所期望的拟合结果：

C#

```
TrendLine tl = new TrendLine();
tl.Label = "Trend";
tl.FitType = FitType.Polynom;
tl.Order = 6;
```



图中的图表使用的数据展示了汽油的价格，这些值每七天获取一次。对于这一类数据，使用MovingAverage趋势线是比较合适的。在下图中你可以看到它是如何拟合数据的：



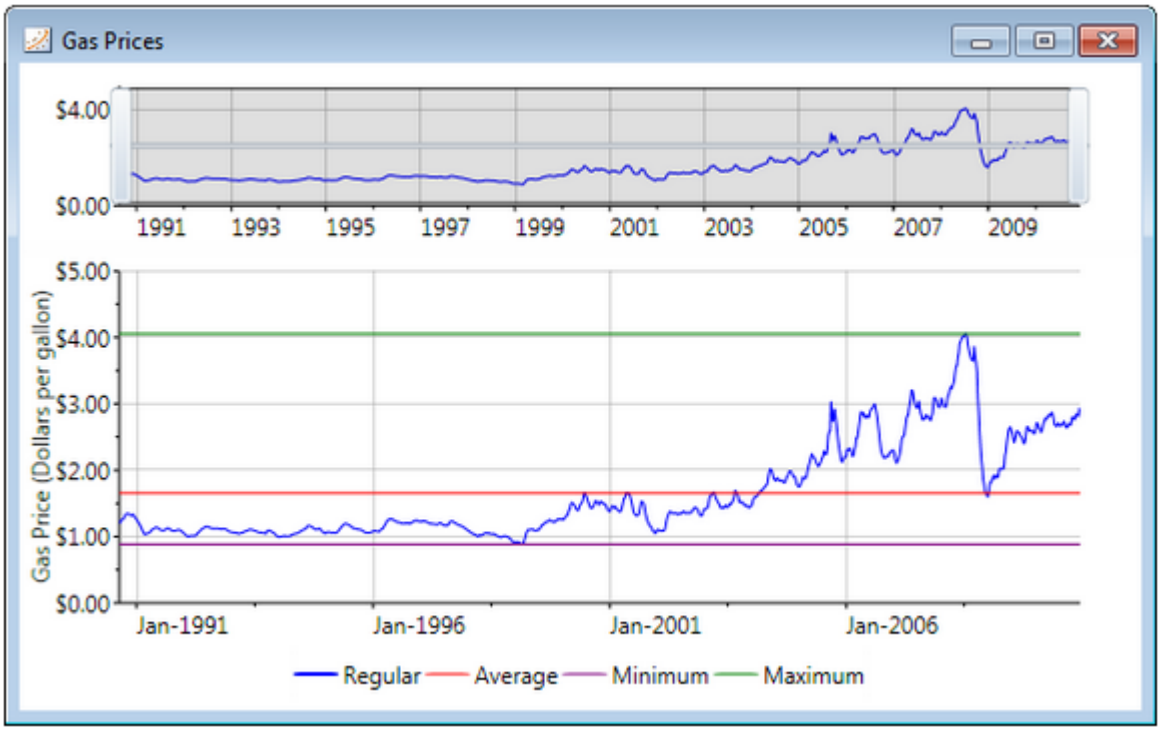
当建立一个移动平均趋势线，你必须实例化一个新的MovingAverage对象并设置Period属性。此属性指定用于趋势线的数据点的个数。由于天然气价格每七天计算一次平均值，每年48个平均值：

C#

```
MovingAverage ma = new MovingAverage();
ma.Label = "Moving Average";
ma.Period = 48;
ma.XValuesSource = days;
ma.ValuesSource = price;
ma.ConnectionStroke = new SolidColorBrush(Colors.Red);
chart.Data.Children.Add(ma);
```

非回归趋势线

如果您希望高亮显示您简单数据，比如说图表数据的最小值，最大值以及平均值，您可以实例化三个趋势线并设置其FitType属性的值为MinmunX，MaximumX以及AverageX：



教程

以下部分介绍如何为图表绑定数据。

数据绑定教程

以下部分包括C1Chart的数据绑定教程。本教程提供一步一步的说明。按照本章中所概述的步骤，你将能够把C1Chart绑定到数据表和XML文件。

以下两个教程的核心属性如下：

- ItemsSource (在线文档 'ItemsSource 属性') - 提供一个对象列表
- ItemNameBinding (在线文档 'ItemNameBinding 属性') -指定一个元素的名称，比如说X轴上的标签。
- ValueBinding (在线文档 'ValueBinding 属性') - 指定数值型值

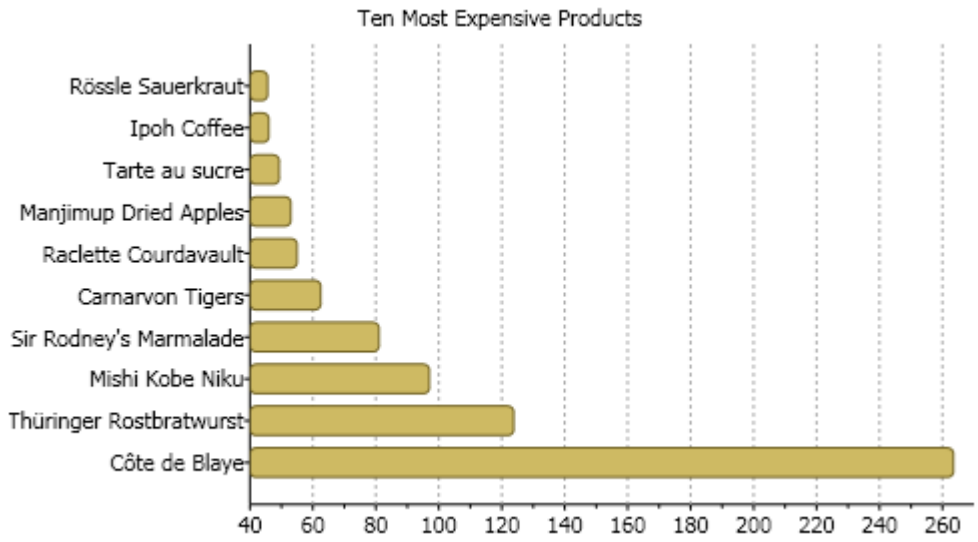
声明式绑定到数据表

注意：本教程适用于WPF应用程序

本教程提供了一步一步的指示绑定C1Chart 到数据集的声明。数据将信息作为简单的条形图显示，有一个y-轴表示产品的名称以及一个x-轴表示每一个产品的单价。产品最贵的十种产品的单价以降序方式显示。条形图通过一个系列绘制单价。没有使用图例，因为仅有一个系列。

图表使用来自于Access的示例数据库，Nwind.mdb。本教程假定数据库文件Nwind.mdb位于Documents\ComponentOne Samples\Common目录，并且为了简洁起见，通过其文件名，而不是其完整路径名引用。

完成本教程将产生一个图表，看起来像以下：



为绑定到C1Chart到数据表声明，完成以下步骤：

1. 在Visual Studio中创建一个新的WPF项目。有关创建WPF项目的更多信息，参见WPF入门。
2. 添加到C1.WPF.C1Chart引用至您的项目。
3. 添加C1Chart控件至窗体。更多信息请参见ComponentOne Studio WPF 版本入门。
4. 一旦C1Chart控制放在窗体上，则将自动生成下面的XAML代码：

XAML

```
Title="Window1" Height="300" Width="500" xmlns:c1chart="clr-
```

```
namespace:C1.WPF.C1Chart;assembly=C1.WPF.C1Chart" Loaded="Window_Loaded">
    <Grid>
        <c1chart:C1Chart Content="" Margin="10,10,10,18" Name="c1Chart1">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:ChartData.ItemNames>P1 P2 P3 P4
P5</c1chart:ChartData.ItemNames>
                    <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25"
/>
                    <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15"
/>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
            <c1chart:Legend DockPanel.Dock="Right" />
        </c1chart:C1Chart>
    </Grid>
```

5. 选择XAML选项卡，添加下面的XAML代码的命名空间：

XAML

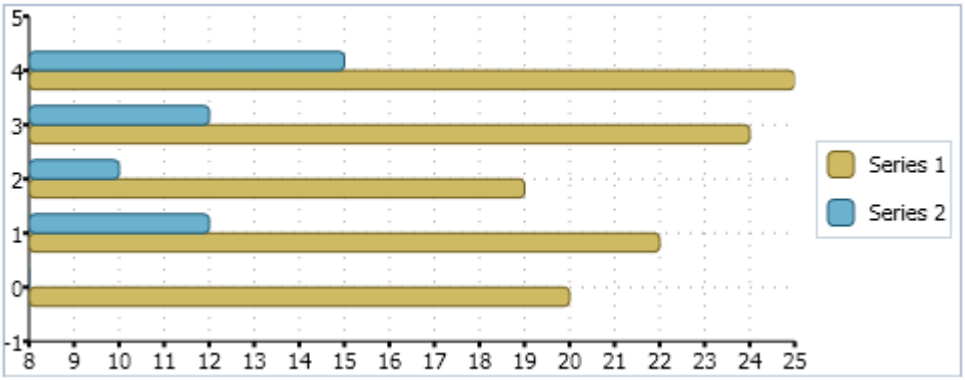
```
 xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

6. 在XAML标记中，更改标题的宽度从300到500。
7. 在<c1chart:C1Chart> 标签内部修改Margin为"0" 并设置ChartType 为 "Bar"。这将改变默认图表的外观从柱形到条形。您的XAML标记应该如下所示：

XAML

```
<Grid>
        <c1chart:C1Chart Margin="0" Name="c1Chart1" ChartType="Bar" Height="185"
VerticalAlignment="Top">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25"
/>
                    <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15"
/>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
            <c1chart:Legend DockPanel.Dock="Right" />
        </c1chart:C1Chart>
        <TextBlock DockPanel.Dock="Top" Text="Ten Most Expensive Products"
HorizontalAlignment="Center"/>
</Grid>
```

您的图表将显示如下：



8. 在c1chart:C1Chart标签结束位置创建一个标签，并将其标签内容设置为“最贵的十种产品”。

XAML

```
<TextBlock DockPanel.Dock="Top" Text="Ten Most Expensive Products"
HorizontalAlignment="Center"/>
```

您的XAML标记现在应该看起来类似以下代码所示：

XAML

```
<Grid>
        <c1chart:C1Chart Margin="0" Name="c1Chart1" ChartType="Bar" Height="185"
VerticalAlignment="Top">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25"
/>
                    <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15"
/>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
            <c1chart:Legend DockPanel.Dock="Right" />
        </c1chart:C1Chart>
        <TextBlock DockPanel.Dock="Top" Text="Ten Most Expensive Products"
HorizontalAlignment="Center"/>
</Grid>
```

9. 在后台代码文件添加以下using/imports代码：

Visual Basic

```
//WPF
Imports System.Data
Imports System.Data.OleDb
Imports C1.WPF.C1Chart
```

C#

```
//WPF
```

```
using System.Data;
using System.Data.OleDb;
using C1.WPF.C1Chart;
```

10. 在Window_Loaded 函数以外为DataSet声明变量，接下来添加以下代码以获取来自于数据库的产品和单价信息：

Visual Basic

```
Private _dataSet As DataSet
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' 创建连接并填充数据集
    Dim mdbFile As String = "c:\Program Files\ComponentOne Studio.NET
2.0\Common\nwind.mdb"
    Dim connString As String = String.Format("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source={0}", mdbFile)
    Dim conn As New OleDbConnection(connString)
    Dim adapter As New OleDbDataAdapter("SELECT TOP 10 ProductName, UnitPrice" &
Chr(13) & "" & Chr(10) & " FROM Products ORDER BY UnitPrice DESC;", conn)
    _dataSet = New DataSet()
    adapter.Fill(_dataSet, "Products")
    ' 设置图表数据的源
    c1Chart1.Data.ItemsSource = _dataSet.Tables("Products").Rows
End Sub
```

C#

```
DataSet _dataSet;
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
      // 创建连接并填充数据集
        string mdbFile = @"c:\Program Files\ComponentOne Studio.NET
2.0\Common\nwind.mdb";
      string connString = string.Format("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source={0}", mdbFile);
      OleDbConnection conn = new OleDbConnection(connString);
  OleDbDataAdapter adapter =
    new OleDbDataAdapter(@"SELECT TOP 10 ProductName, UnitPrice
    FROM Products ORDER BY UnitPrice DESC;", conn);
      _dataSet = new DataSet();
      adapter.Fill(_dataSet, "Products");

      // 设置图表数据的源
      c1Chart1.Data.ItemsSource = _dataSet.Tables["Products"].Rows;
    }
```

注意：请确保mdbFile文件的路径匹配nwind.mdb数据库文件在您本机的实际位置。

11. 在XAML选项卡以切换至XAML视图，接下来删除以下来自于ChartData的默认数据：

XAML

```
<c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>
```

```
        <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25" />
        <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15" />
```

现在C1Chart控件在窗体上显示为空白。

12. 在<c1chart:C1Chart.Data>标签内部添加ItemNameBinding 至ChartData以指定元素的名称，也就是显示在y-轴上的标签，同时设置ValueBinding属性至DataSeries，以便为系列指定数值类型的值。

XAML

```
<c1chart:ChartData ItemNameBinding="{Binding Path=[ProductName]}">
        <c1chart:DataSeries ValueBinding="{Binding Path=[UnitPrice]}"/>
</c1chart:ChartData>
```

您工程中的XAML代码应当看起来如以下所示：

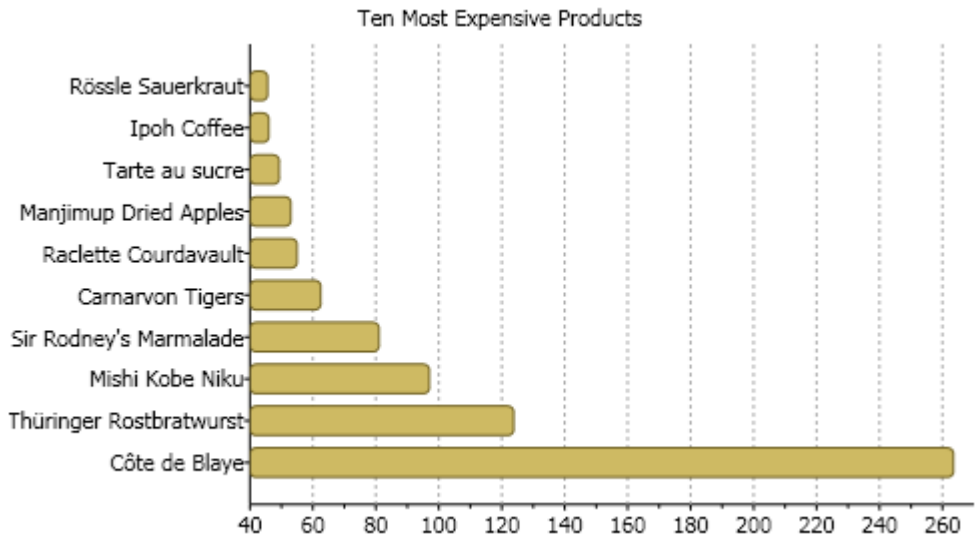
XAML

```
<Window x:Class="Chart for WPF_QuickStart.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    Title="Window1" Height="300" Width="500" Loaded="Window_Loaded"
xmlns:c1chart="clr-namespace:C1.WPF.C1Chart;assembly=C1.WPF.C1Chart">
    <Grid>
        <c1chart:C1Chart Margin="0" Name="c1Chart1" ChartType="Bar">
            <TextBlock DockPanel.Dock="Top" Text="Ten Most Expensive Products"
                 HorizontalAlignment="Center"/>
            <c1chart:C1Chart.Data>
                <c1chart:ChartData ItemNameBinding="{Binding Path=
[ProductName]}">
                    <c1chart:DataSeries ValueBinding="{Binding Path=
[UnitPrice]}"/>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
        </c1chart:C1Chart>
    </Grid>
</Window>
```

13. 从XAML中删除<c1chart:Legend DockPanel.Dock="Right" />标签，以便删除内置的图例控件。
14. 运行您的工程，以确保一切正常工作。

在运行时可以看到以下效果

图表显示来自产品表的数据。



绑定到一个XML

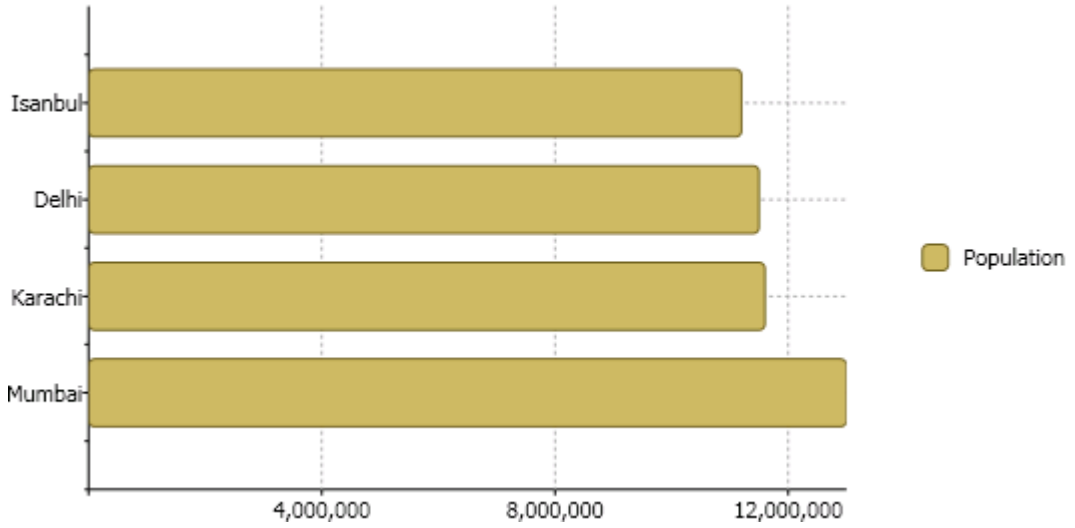
本教程提供了一个将XML作为数据到嵌入到XAML页面以绑定C1Chart控件至该XML数据的分步说明。该数据将信息显示为一个简单的条形图，y-轴表示城市的名称，x-轴表示每一个城市的人口。该条形图使用一个系列绘制人口信息。图例区用于显示人口的颜色。

本教程中的绑定设置在ChartData类上，通过以下XAML代码：

XAML

```
<c1chart:ChartData ItemsSource="{Binding Source={StaticResource data}}"
   ItemNameBinding="{Binding XPath=CityName}">
     <c1chart:DataSeries Label="Population"
       ValueBinding="{Binding XPath=Population}" />
</c1chart:ChartData>
```

完成本教程将产生一个图表，看起来像以下：



绑定 C1Chart 至XML：

1. 在Visual Studio中创建一个新的WPF工程。有关创建WPF项目的更多信息，参见WPF入门。
2. 在资源部分中，将数据直接嵌入到数据岛。XML数据岛必须包含在在<x:Xdata>标签中，并且总是有一个根节点，在本示例中，该根节点为Cities：

XAML

```
<Grid.Resources>
      <XmlDataProvider x:Key="data" XPath="Cities/City">
        <x:XData>
          <Cities xmlns="">
            <City>
              <CityName>Mumbai</CityName>
              <Population>13000000</Population>
            </City>
            <City>
              <CityName>Karachi</CityName>
              <Population>11600000</Population>
            </City>
            <City>
              <CityName>Delhi</CityName>
              <Population>11500000</Population>
            </City>
            <City>
              <CityName>Istanbul</CityName>
              <Population>11200000</Population>
            </City>
          </Cities>
        </x:XData>
      </XmlDataProvider>
</Grid.Resources>
```

3. 向您的工程添加到 C1.WPF.C1Chart 的引用。
4. 添加 C1Chart控件窗体。

一旦C1Chart控件添加到窗体，将会生成以下XAML代码：

XAML

```
Title="Window1" Height="50" Width="100" xmlns:c1chart="clr-
namespace:C1.WPF.C1Chart;assembly=C1.WPF.C1Chart" Loaded="Window_Loaded">
    <Grid>
        <c1chart:C1Chart Content="" Margin="10,10,10,18" Name="c1Chart1">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData>
                    <c1chart:ChartData.ItemNames>P1 P2 P3 P4
P5</c1chart:ChartData.ItemNames>
                    <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25"
/>
                    <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15"
/>
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
            <c1chart:Legend DockPanel.Dock="Right" />
        </c1chart:C1Chart>
    </Grid>
```

5. 改变窗体的宽度为 "300" ，高度为 "550"
6. 在<c1chart:C1Chart> 标签内部修改Margin为"0" 并设置ChartType 为 "Bar"。这将改变默认图表的外观从柱形到条形。你的XAML代码应该看起来如以下所示：

XAML

```
<c1chart:C1Chart Margin="0" Name="c1Chart1" ChartType="Bar">

</c1chart:C1Chart>
```

7. 在XAML文件中找到<c1chart:C1Chart.Data>标签，并从中删除以下XAML代码：

XAML

```
<c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>
        <c1chart:DataSeries Label="Series 1" Values="20 22 19 24 25" />
        <c1chart:DataSeries Label="Series 2" Values="8 12 10 12 15" />
```

两个默认的系列从C1Chart移除，现在C1Chart显示为空白，因为它没有任何数据。

8. 在<c1chart:C1Chart.Data>标签内部，向ChartData 添加ItemNameBinding属性以指定元素的名称，在本示例中，这是显示在y-轴上的标签，同时指定DataSeries的ValueBinding属性以指定系列的数值型的值。以下实例通过绑定扩展绑定ChartData.ItemsSource属性，指定数据源。ChartData.ItemNameBinding属性通过使用绑定扩展指定Path进行绑定。DataSeries.Label属性通过使用绑定扩展进行绑定，指定Path，在这里是Population。

XAML

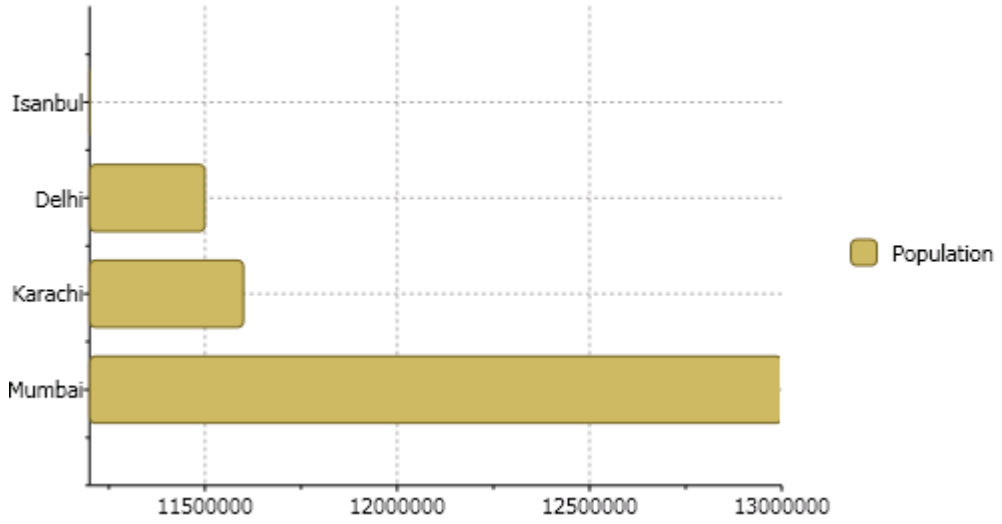
```
<c1chart:ChartData ItemsSource="{Binding Source={StaticResource data}}"
                       ItemNameBinding="{Binding XPath=CityName}">
          <c1chart:DataSeries Label="Population" ValueBinding="{Binding
XPath=Population}" />
</c1chart:ChartData>
```

您的C1Chart的XAML代码应当看起来如下所示：

XAML

```
<c1chart:C1Chart Height="300"  HorizontalAlignment="Left" Margin="0"
Name="c1Chart1" ChartType="Bar" VerticalAlignment="Top" Width="500">
            <c1chart:C1Chart.Data>
                <c1chart:ChartData ItemsSource="{Binding Source={StaticResource
data}}"
                      ItemNameBinding="{Binding XPath=CityName}">
                    <c1chart:DataSeries Label="Population"
                        ValueBinding="{Binding XPath=Population}" />
                </c1chart:ChartData>
            </c1chart:C1Chart.Data>
            <c1chart:Legend DockPanel.Dock="Right" />
</c1chart:C1Chart>
```

9. 运行您的工程，以确保一切正常工作。
   您的图表将看起来如下所示：



请注意这里x-轴的标注是如何显示的。我们需要格式化x-轴的标注使得人口的数值显示千位分隔符。

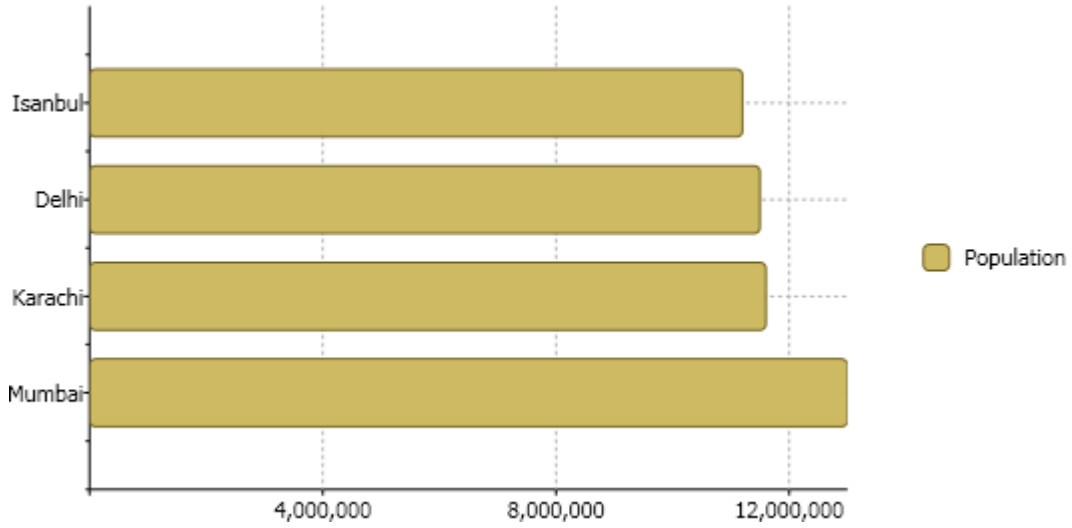
10. 为C1Chart的ChartView.AxisX属性声明标签。您将需要设置AxisX 的以下属性的以格式化标注以及网格线。

    在结束标签</c1chart:C1Chart.Data>之后添加以下XAML代码：

XAML

```
<c1chart:C1Chart.View>
        <c1chart:ChartView>
          <c1chart:ChartView.AxisX >
            <c1chart:Axis Min="0" MajorGridStroke="DarkGray"
AnnoFormat="#,###,###"/>
          </c1chart:ChartView.AxisX>
        </c1chart:ChartView>
</c1chart:C1Chart.View>
```

图表上的x-轴的标注将更新显示为以下：



使用MVVM

WPF及Silverlight版Chart支持MVVM（模型-视图-ViewModel）设计模式。整个图表可以被声明式地通过源生的绑定技术绑定到XAML。

下面的步骤演示了如何在一个MVVM架构的WPF或Silverlight应用程序中使用C1Chart（在线文档 'C1Chart 类'）。

1. 创建一个新类命名为Sale，实现INotifyPropertyChanged接口。

C#

```
public class Sale : INotifyPropertyChanged
{
    private string _product;
    private double _value;
    private double _discount;
    public Sale(string product, double value, double discount)
    {
        Product = product;
        Value = value;
        Discount = discount;
    }
    public string Product
    {
        get { return _product; }
        set
        {
            if (_product != value)
            {
                _product = value;
                OnPropertyChanged("Product");
            }
        }
    }
    public double Value
```

```
    {
        get { return _value; }
        set
        {
            if (_value != value)
            {
                _value = value;
                OnPropertyChanged("Value");
            }
        }
    }
    public double Discount
    {
        get { return _discount; }
        set
        {
            if (_discount != value)
            {
                _discount = value;
                OnPropertyChanged("Discount");
            }
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

2. 创建一个新的类并命名为SaleViewModel。这将做为包含C1Chart的View的数据上下文。

C#

```
public class SaleViewModel : INotifyPropertyChanged
{
    private ObservableCollection<Sale> _sales = new ObservableCollection<Sale>
();

    public SaleViewModel()
    {
        //加载数据
        LoadData();
    }
    public ObservableCollection<Sale> Sales
    {
        get { return _sales; }
    }
    public void LoadData()
    {
        //TODO: 从数据源加载数据
```

```
        _sales.Add(new Sale("Bikes", 23812.89, 12479.44));
        _sales.Add(new Sale("Shirts", 79752.21, 19856.86));
        _sales.Add(new Sale("Helmets", 63792.05, 16402.94));
        _sales.Add(new Sale("Pads", 30027.79, 10495.43));
    }
    public event PropertyChangedEventHandler PropertyChanged;
    void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

该类将包含一个ObservableCollection，Sales，以及生成模拟数据的初始化方法。

3. 切换到源代码视图并创建一个新的WPF或者Silverlight的用户控件，并命名为 SaleView.xaml。在LayoutRoot Grid之前添加以下XAML：

XAML

```
<UserControl.Resources>
    <local:SaleViewModel x:Key="viewModel" />
</UserControl.Resources>
<UserControl.DataContext>
    <Binding Source="{StaticResource viewModel}"/>
</UserControl.DataContext>
```

该XAML声明了一个SalesViewModel做为资源，并设置其为UserControl的DataContext。在运行时，View将被绑定到ViewModel。在View中的控件可以绑定到ViewModel的公共属性。

4. 向页面添加一个C1Chart控件。
5. 替换C1Chart默认的XAML为以下代码：

XAML

```
<c1:C1Chart ChartType="Column" Name="c1Chart1" Palette="Module">
    <c1:C1Chart.Data>
        <c1:ChartData ItemsSource="{Binding Sales}" ItemNameBinding="{Binding
Product}">
            <c1:DataSeries Label="Value" ValueBinding="{Binding Value}" />
            <c1:DataSeries Label="Discount" ValueBinding="{Binding Discount}" />
        </c1:ChartData>
    </c1:C1Chart.Data>
    <c1:C1ChartLegend />
</c1:C1Chart>
```

该XAML定义了一个具有两个数据系列的C1Chart。该ChartData的ItemsSource设置为一个Sales对象的集合，该集合暴露在ViewModel上。每个DataSeries 都有其ValueBinding 属性设置并设置ItemNameBinding 以沿x轴展示我们的产品名称。

> **注意**：如果你使用的是XYDataSeries，那么你应该指定每个系列的XValueBinding，而不应该设置ItemNameBinding。

5. 打开App.xaml.cs应用程序配置文件并将Application_Startup 事件，替换为以下代码：

Example Title

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    this.RootVisual = new Views.SaleView();
}
```

该代码设置RootVisual以便在启动时显示SalesView。

6. 运行应用程序并观察到C1Chart 已经绑定到ViewMode上的Sales数据。