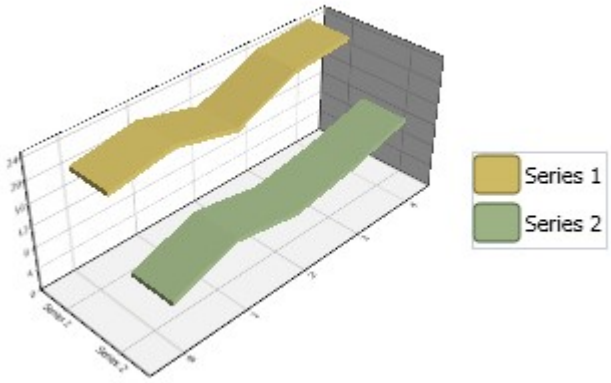
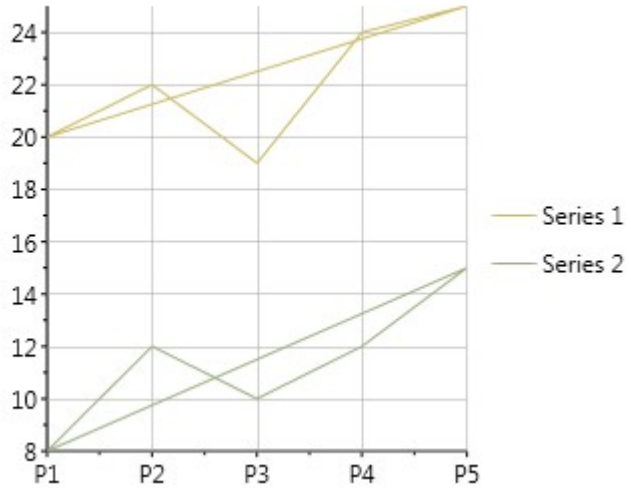
3D Ribbon Chart

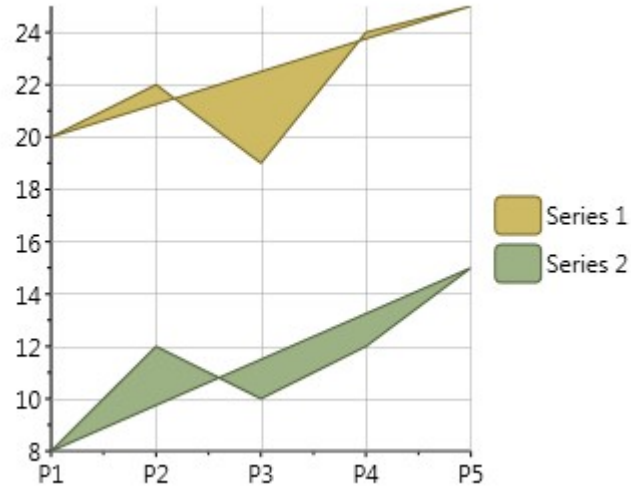
这个图表类型只在WPF中可用。
下面的图像展示了当您设置ChartType属性为Ribbon之后三维带状图的外观：



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：

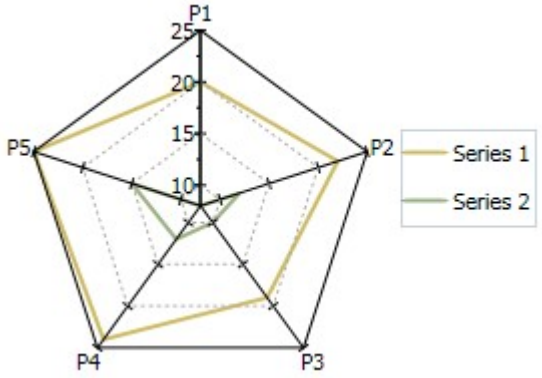


下图展示了当您设置ChartType（在线文档 'ChartType属性'）属性的值为PolygonFilled时，多边形填充图的外观：

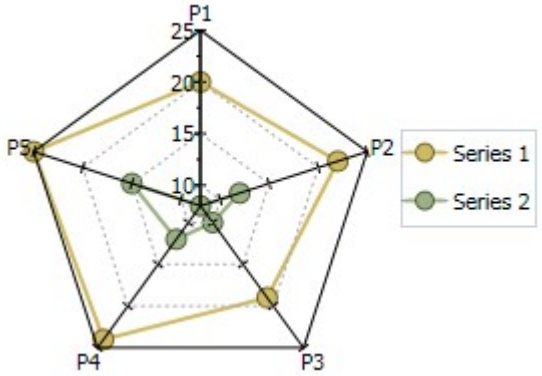


雷达图
雷达图是一个极坐标图的变体。雷达图沿着雷达线方向绘制每一个数据集的Y值。如果数据具有N个独一无二的电，则图表平面将被分成N个夹角相等的段，且每隔360/N度的增量绘制一条雷达线（每条雷达线表示每一个数据点）。默认情况下，第一个点的雷达线垂直方向绘制（90度）。
雷达图的标签可以通过ChartData.ItemNames（在线文档 'ItemNames属性'）属性进行设置。这些标签位于每个径向线的末端。
设置起始角度设置起始角度
PolarRadarOptions（在线文档 'PolarRadarOptions 类'）类的PolarRadarOptions.SetStartingAngle（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

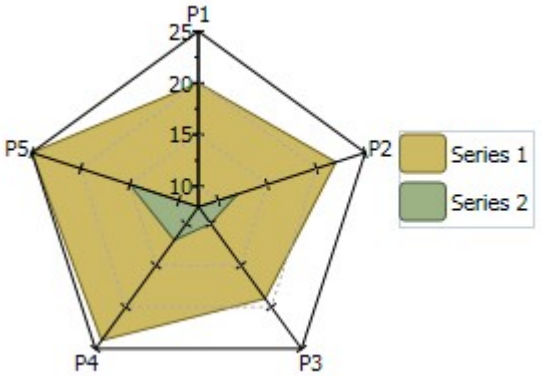
SetStartingAngle属性的值为90，那么图表将沿着顺时针方向旋转90度。
下图显示当您设置ChartType属性的值为Radar时雷达图的外观。



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。

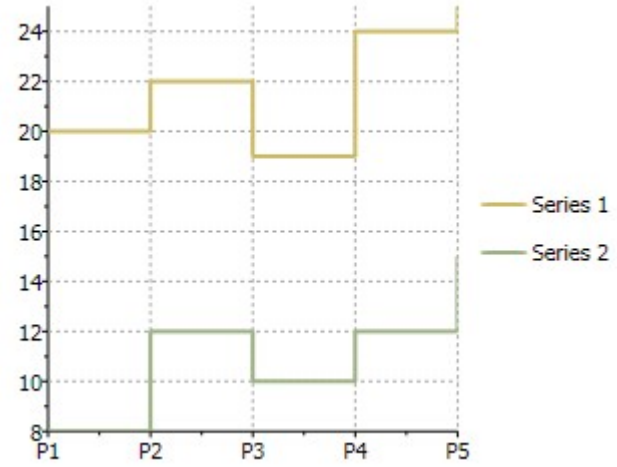


下图显示当您设置ChartType属性的值为RadarFilled时填充雷达图的外观。

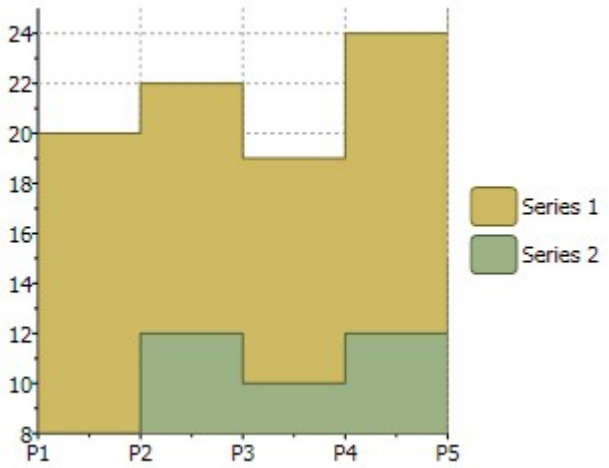


阶梯图

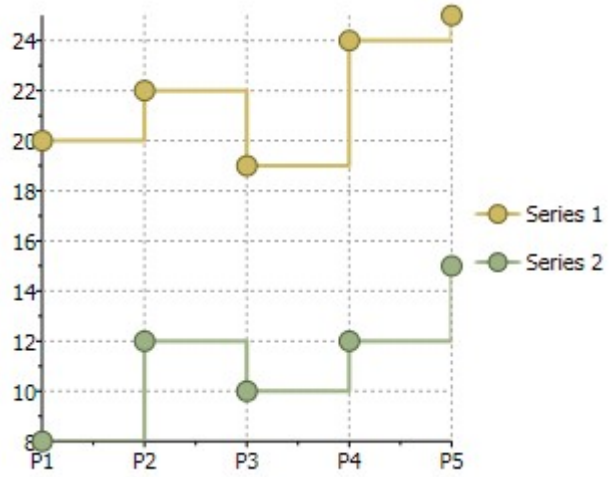
阶梯图是XY图的一种。阶梯图通常用于Y值发生离散的改变，且在某个特定的X值位置发生了一个突然的变化。
一个简单的，日常例子是按照时间记录的支票簿的曲线。当每一次存款发生，以及每一次书写支票支出时，支票登记的余额（Y值）都是立即发生变更，而不是按照时间的推移，缓缓的改变。在没有存入款项或书写支票的时候，余额（Y 值）随着时间的流逝保持不变。
和折线图以及XY图类似，阶梯图的外观可以通过每一个系列的Connection以及Symbol属性进行自定义，包括修改颜色，符号尺寸以及线形的粗细。
符号可以完全删除，以强调点之间的关系或包含到表示实际的数据值。如果存在数据空洞，阶梯图仍然可以按照预期表现，并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：



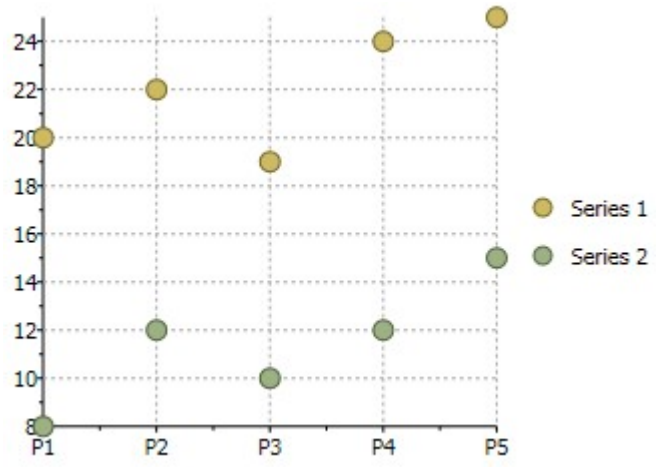
下图显示当您设置ChartType属性的值为StepSymbols时，符号阶梯图的外观：



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。
步骤步骤1）选择图表类型：）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;
```

步骤步骤2）设置坐标轴：）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
```

步骤步骤3）添加一个或多个数据系列）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);
```

步骤步骤4）调整图表的外观）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#

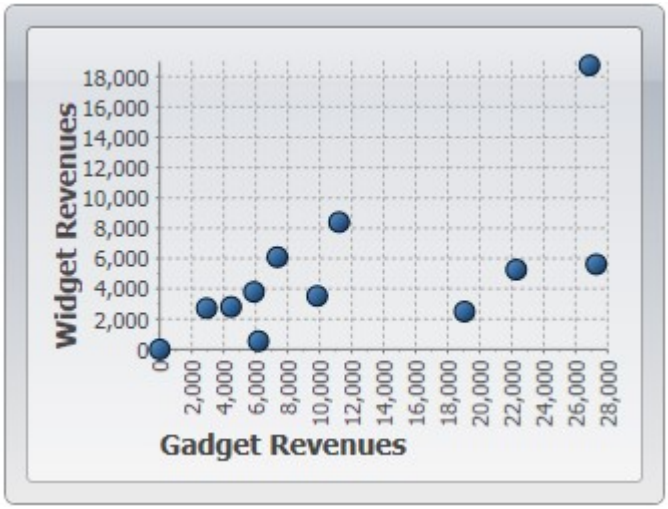
```
c1Chart.Theme = c1Chart.TryFindResource(new
ComponentResourceKey(typeof(C1.WPF.C1Chart.C1Chart), "Office2007Black")) as

ResourceDictionary;
}
```

Silverlight

C#

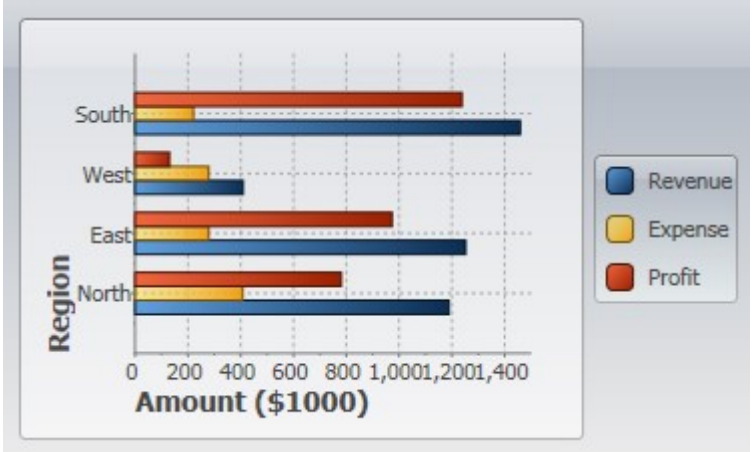
```
c1Chart.Theme = Theme.Office2007Black; }
```

完成的图表将显示完成的图表将显示Gadget 和和Widget的收入之间的正相关关系。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：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 dou
nue { get; set; }
```

```
<span style="color: #0000ff">public</span> <span style="color: #0000ff">double</span> Expense { <span style="color:
#0000ff">get</span>; <span style="color: #0000ff">set</span>; } <span style="color: #0000ff">public</span> <span
style="color: #0000ff">double</span> Profit { <span style="color: #0000ff">get</span> { <span style="color:
#0000ff">return</span> Revenue - Expense; } }
<span style="color: #008000">// 构造器一</span>
<span style="color: #0000ff">public</span> SalesRecord(<span style="color: #0000ff">string</span> region, <span style="color:
#0000ff">double</span> revenue, <span style="color: #0000ff">double</span> expense)
{
Region = region;
Revenue = revenue;
Expense = expense;
}
<span style="color: #008000">// 构造器二</span>
<span style="color: #0000ff">public</span> SalesRecord(DateTime month, <span style="color: #0000ff">string</span> product,
<span style="color: #0000ff">double</span> revenue, <span style="color: #0000ff">double</span> expense) {
Date = month;
Product = product;
Revenue = revenue;
Expense = expense;
}
}
<span style="color: #008000">// 返回一个列表，每一个区域一个SalesRecord记录</span>
List<SalesRecord> GetSalesPerRegionData() {
<span style="color: #0000ff">var</span> data = <span style="color: #0000ff">new</span> List<SalesRecord>(); Random rnd =
<span style="color: #0000ff">new</span> Random(0); <span style="color: #0000ff">foreach</span> (<span style="color:
#0000ff">string</span> region <span style="color: #0000ff">in</span> <span style="color:
#800000">"North,East,West,South"</span>.Split(<span style="color: #800000">','</span>))
{ data.Add(<span style="color: #0000ff">new</span> SalesRecord(region, 100 + rnd.Next(1500), rnd.Next(500)));
} <span style="color: #0000ff">return</span> data;
}
<span style="color: #008000">// 返回一个列表，每一个产品一条SalesRecord记录</span>
<span style="color: #008000">// 在过去的12个月期间内</span>
List<SalesRecord> GetSalesPerMonthData()
{
<span style="color: #0000ff">var</span> data = <span style="color: #0000ff">new</span> List<SalesRecord>(); Random rnd =
<span style="color: #0000ff">new</span> Random(0); <span style="color: #0000ff">string</span>[] products = <span
style="color: #0000ff">new</span> <span style="color: #0000ff">string</span>[] {<span style="color:
#800000">"Widgets"</span>, <span style="color: #800000">"Gadgets"</span>, <span style="color: #800000">"Sprockets"</span> };
<span style="color: #0000ff">for</span> (<span style="color: #0000ff">int</span> 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：步骤步骤1）选择图表类型：）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;

}
```

步骤步骤2）设置坐标轴：）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 ";
```

步骤步骤3）添加一个或多个数据系列）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).ToAr
ray(); 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。
步骤步骤4）调整图表的外观）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;}
```

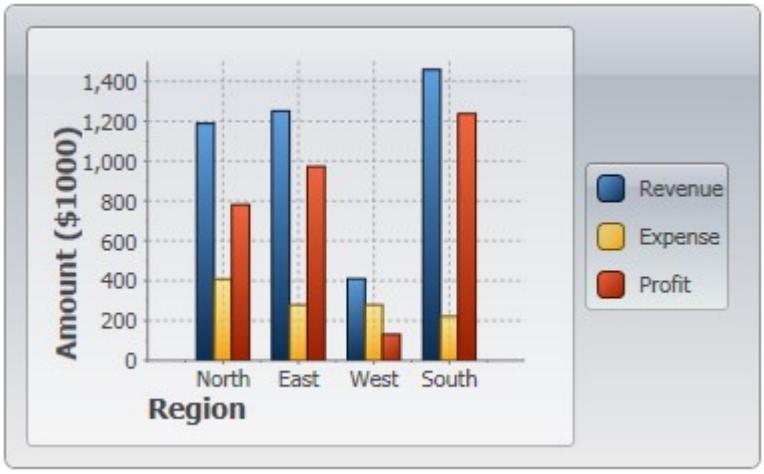
C#// 设置主题c1Chart.Theme = _c1Chart.TryFindResource(new ComponentResourceKey(typeof(C1.Silverlight.C1Chart.C1Chart),"Office2007Black")) as ResourceDictionary;}

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：

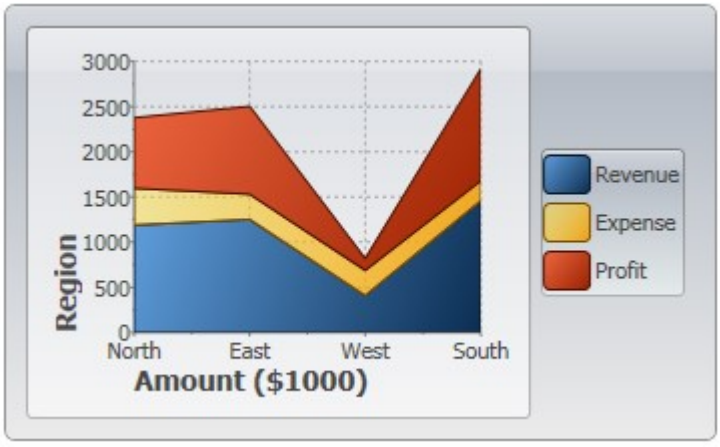
C#

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

}
```

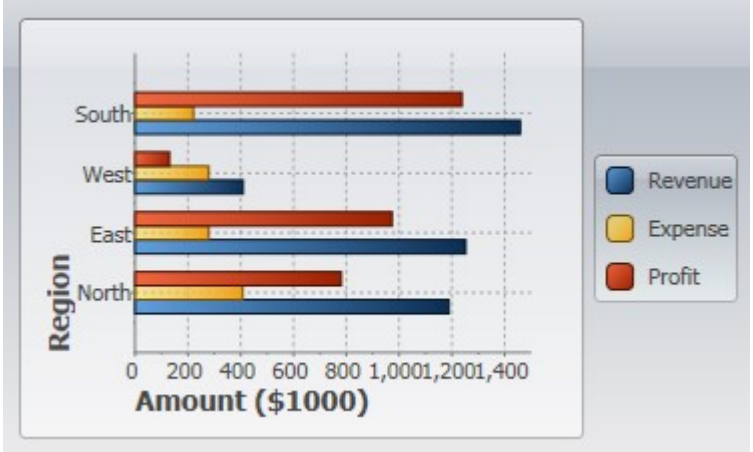
以上代码将生成简单的数值图表。你可以通过修改ChartType属性为任意余下的简单的图表类型值的以尝试效果：Bar，AreaStacked，Pie以创建不同类型的图表。注意，如果你改变ChartType为Column，您将需要在Y轴上显示数据的标签，所以你会用到AxisY。
其结果应该是类似于下面的图像：
ChartType.Column



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, <ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="ce8d069b-8b43-4d94-91e3-4a9868133859"><ac:plain-text-body><![CDATA[XValuesSource = new double[] {1,2,3 }, ValuesSource = new double[] {1,2,3 } }); chart.Data.Children.Add(new XYDataSeries() {	]]></ac:plain-text-body></ac:structured-ma

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 = 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)
{
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="8f360e8f-2b0b-4cdd-a1ad-8826df2a83e5"><ac:plain-text-body><![CDATA[ // 点的个数 int cnt = 200; var xvals = ne
w double[cnt]; var yvals = new double[cnt]; double dx = (x2 - x1) / (cnt-1);
]]></ac:plain-text-body></ac:structured-macro>
for (int i = 0; i < cnt; i++)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="b3a56b0e-3c8e-4026-8217-054dc894627b"><ac:plain-text-body><![CDATA[ { var x = x1 + dx * i; xvals[i] = x; x =
(x - b) / c; yvals[i] = a * Math.Exp(-0.5*x*x);
]]></ac:plain-text-body></ac:structured-macro>
}
```

```
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();
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="da591002-9c2f-4e12-acda-22639091b005"><ac:plain-text-body><![CDATA[ // 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 }); };
]]></ac:plain-text-body></ac:structured-macro>
```

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

<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。

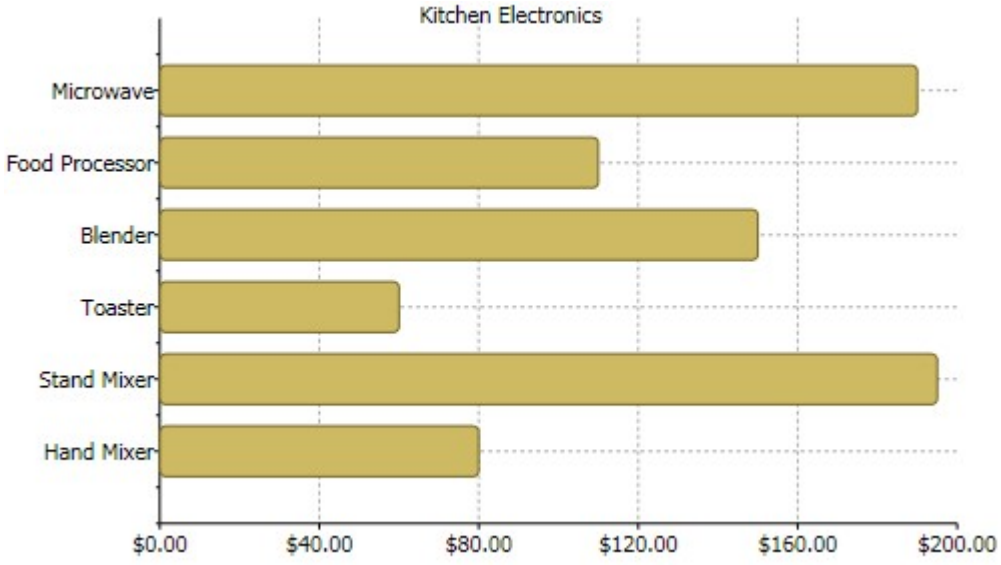
坐标轴

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

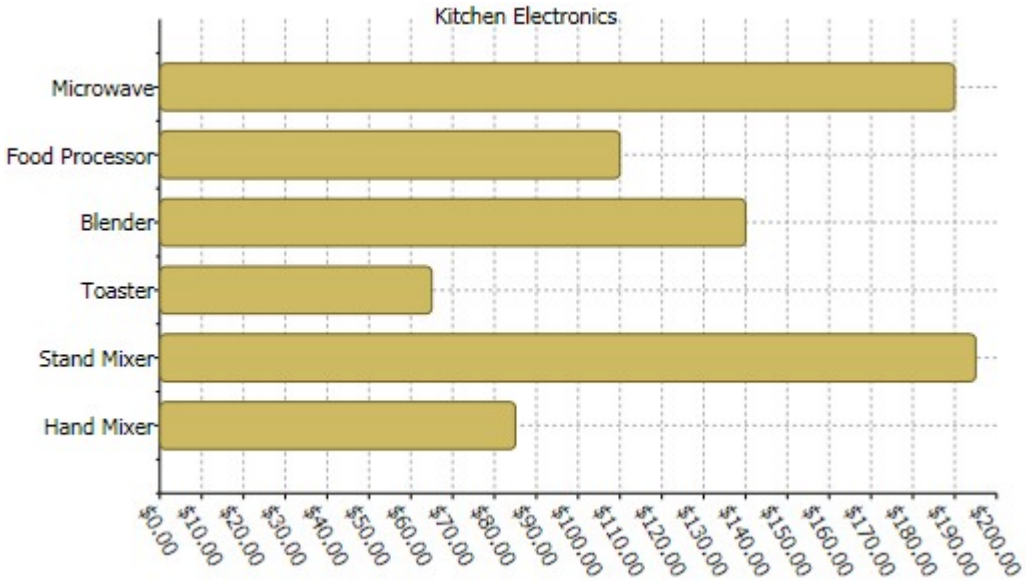
```
<c1chart:C1Chart.View>
<c1chart:ChartView>
<c1chart:ChartView.AxisX>
<c1chart:Axis Min="0" AnnoFormat="c" AutoMin="false"

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#

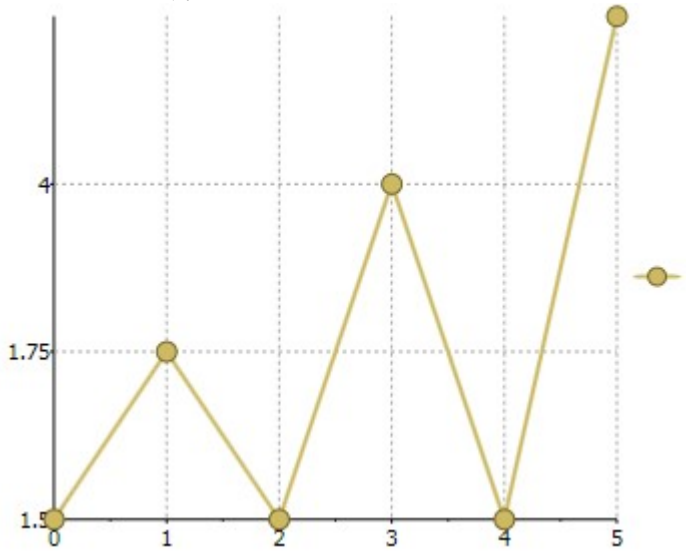
```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="a97b4d2d-61e9-4c43-add9-913e9c0001ae"><ac:plain-text-body><![CDATA[
```

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

```
]]></ac:plain
```

```
}); c1Chart1.ChartType = ChartType.LineSymbols;
c1Chart1.View.AxisY.ItemsSource = <span style="color: #0000ff">new</span> <span style="color: #0000ff">double</span>[] {
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#

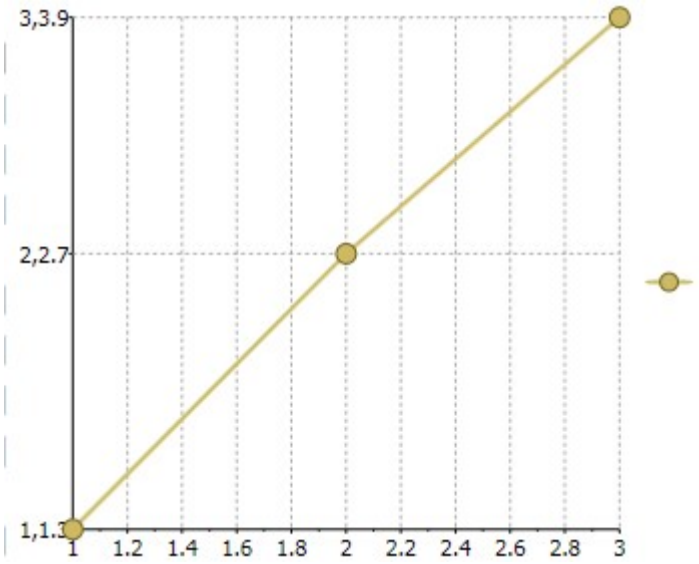
```
c1Chart1.Reset(true);
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="ff8f4ef4-afcf-409c-84db-826fe1e6a519"><ac:plain-text-body><![CDATA[ c1Chart1.Data.Children.Add( new DataSeries() { ValuesSource = new double[] { 1, 2, 1, 3, 1, 4 }
]]></ac:plain-text-body></ac:structured-macro>
}); 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：
!worddav9e309095f0764683f086094b777d4eab.png|height=260,width=348!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);
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="9b6c127a-490e-4612-a590-d10ec3c3f92c"><ac:plain-text-body><![CDATA[ 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;
]]></ac:plain-text-body></ac:structured-macro>
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 p
arameter, 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,
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="ef234d1e-a580-4416-8d43-fdf17be41f4c"><ac:plain-text-body><![CDATA[ ItemsSource = new string[] { ""},
]]></ac:plain-text-body></ac:structured-macro>
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

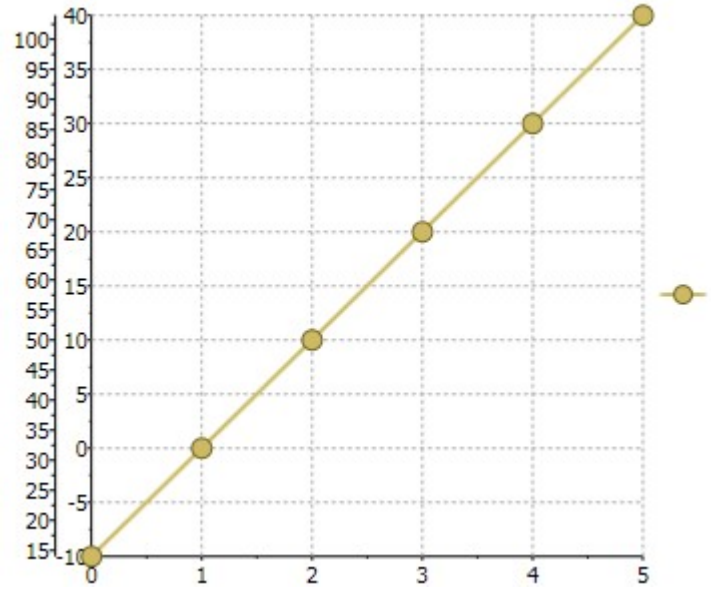
IsDependent （在线文档 'IsDependent 属性'）属性允许链接辅助坐标轴到某一个主坐标轴（AxisX或AxisY，取决于
AxisType）。副坐标轴轴始终具有和主坐标轴相同的最小值和最大值。
新的属性 DependentAxisConverter （在线文档
'DependentAxisConverter属性'）以及委托Axis.AxisConverter指定用来从主坐标轴到副坐标轴坐标系的转换。
下面的代码创建一个副Y轴：

C#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="7ee6479a-b392-4925-80c3-32e4f6364830"><ac:plain-text-body><![CDATA[
```

```
c1Chart1.Reset(true);
c1Chart1.Data.Children.Add( new DataSeries
{ ValuesSource = new double[] { -10, 0, 10
20, 30, 40 }
]]></ac:plain-text-body></ac:structured-ma
});
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()
{
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="90e3d603-06c7-4ff1-8b79-659e1b504007"><ac:plain-text-body><![CDATA[ ValuesSource = new double[] { -1, 2, 0,
2, -2 }, XValuesSource = new double[] { -2, -1, 0, 1, 2 }
]]></ac:plain-text-body></ac:structured-macro>
}; 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" };
```

使用使用XAML代码设置坐标轴标题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

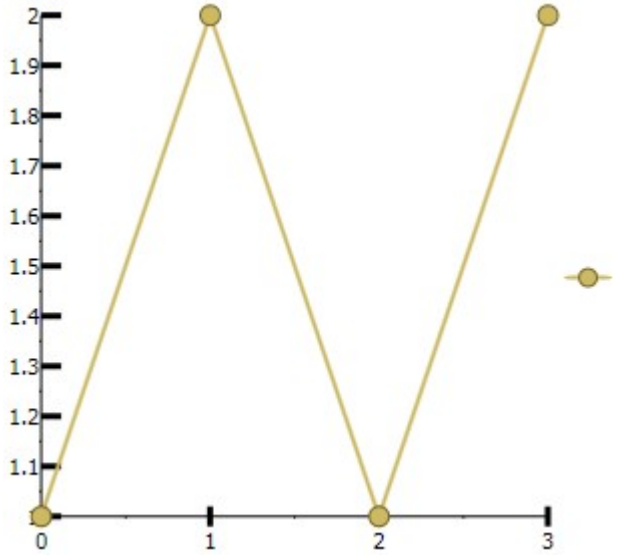
您可以通过为MajorTickOverlap（在线文档 'MajorTickOverlap 属性'）属性指定一个从 0 到 1 的值，决定主刻度线标记的重叠程度。
默认值是0，这意味着没有重叠。当重叠是1，整个刻度线位于绘图区域内。对于 X-轴，当年您增加MajorTickOverlap
属性的值时，刻度线向上移动，而减少该属性的值时，刻度线向下移动。对于
Y-轴，当您增加MajorTickOverlap属性的值时，刻度线向左移动。

C#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="9f602ba0-08d5-43e9-9a44-c5f8c96a843d"><ac:plain-text-body><![CDATA[
```

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



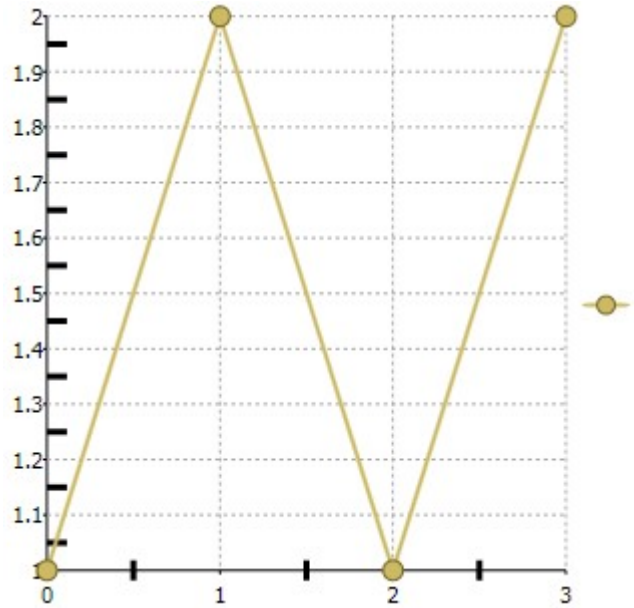
次要刻度线重叠

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#

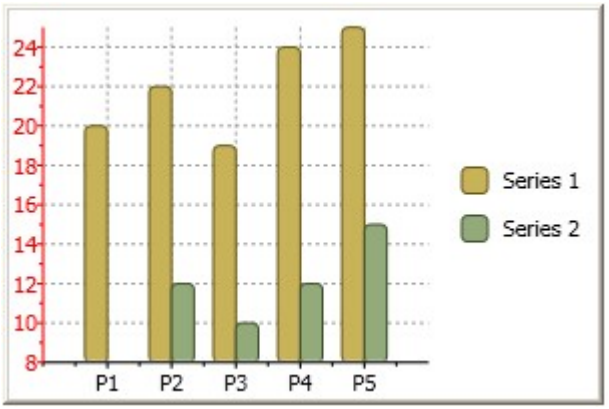
```
c1Chart1.Reset(true);
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="bd5abf87-39d7-45ea-8cd1-ddbc48bf2e7b"><ac:plain-text-body><![CDATA[c1Chart1.Data.Children.Add( new DataSeries
() { ValuesSource = new double[] { 1, 2, 1, 2 } });
]]></ac:plain-text-body></ac:structured-macro>
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。默认刻度线默认刻度线
下图显示默认的刻度线：



自定义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

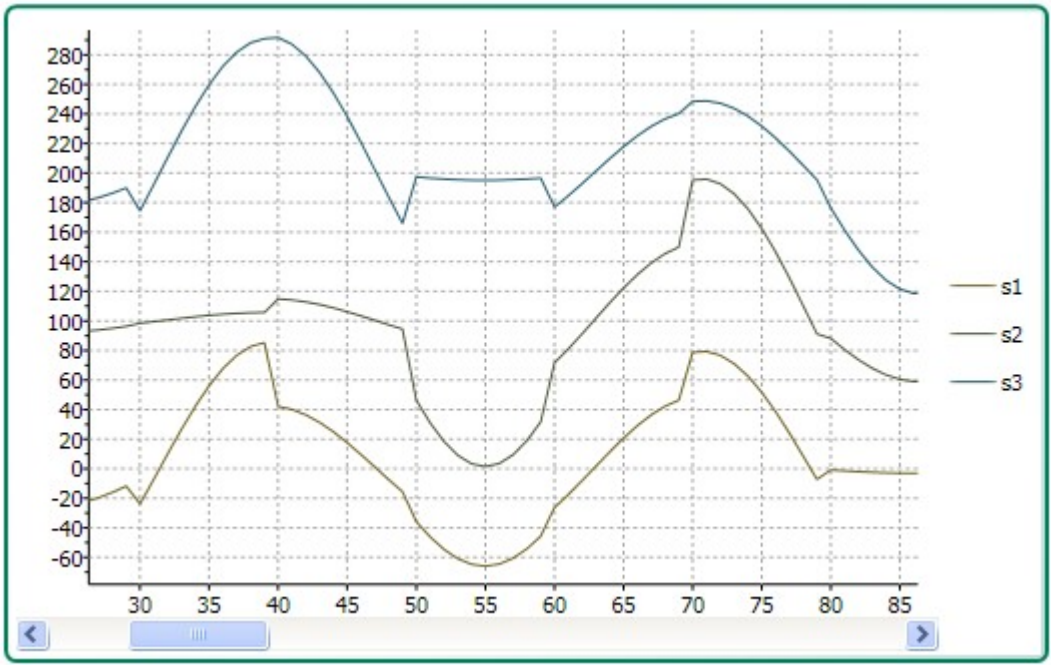
```
C#
c1Chart1.View.AxisY.MajorUnit = 5; c1Chart1.View.AxisY.MinorUnit = 1;
```

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。
绘制或填充主要绘制或填充主要/次要网格线次要网格线
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。
设置主要设置主要/次要网格线的虚线样式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。
设置主要设置主要/次要网格线的线宽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

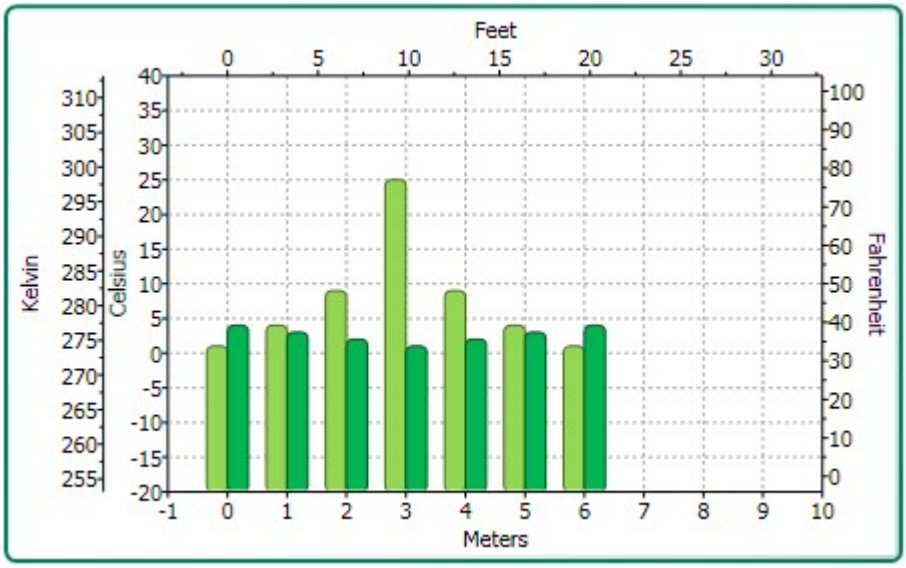
当一个数据集包含某个大范围的某个X值或Y值时，有时正常的图表设置并不能显示最有效的信息。格式化一个图表，使其沿着垂直方向显示Y轴，并且坐标轴标注从最小值开始，反转或逆向显示坐标轴有时可以在视觉上更具有吸引力。因此，C1Chart（在线文档'C1Chart类'）提供了Inverted属性以及Reversed（在线文档 'Reversed 属性'）属性给坐标轴。设置ChartView（在线文档'ChartView类'）的Reversed（在线文档 'Reversed 属性'）属性为True将逆向显示坐标轴。
这意味着，在坐标轴的最大值一侧将被坐标轴最小值一侧所取代，而最小值一侧也将被最大值一侧取代。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

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

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

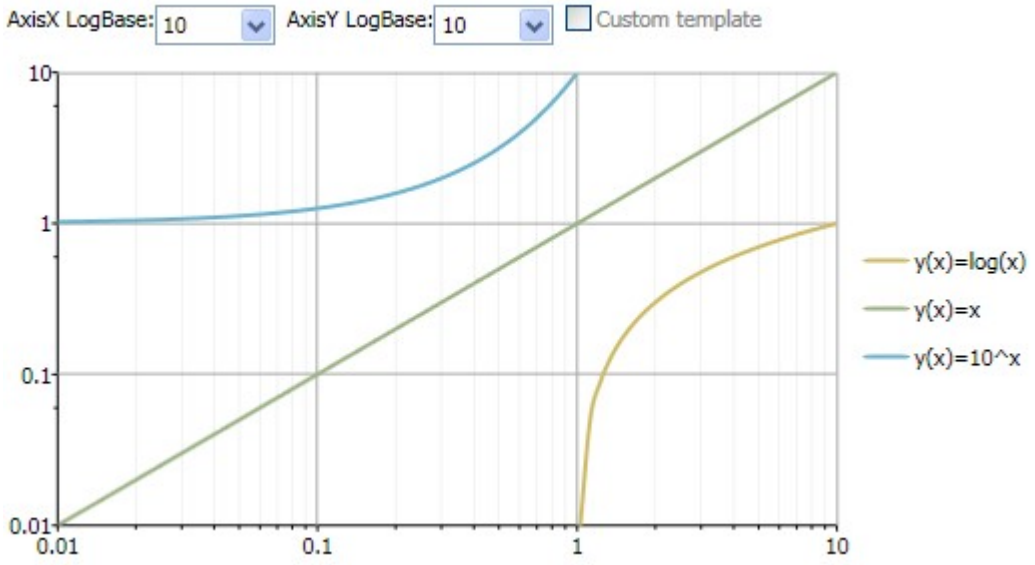
常用对数 – 采用 10 作为基数，因此 log 100 = 2。

自然对数 – 使用数学常数e为基。

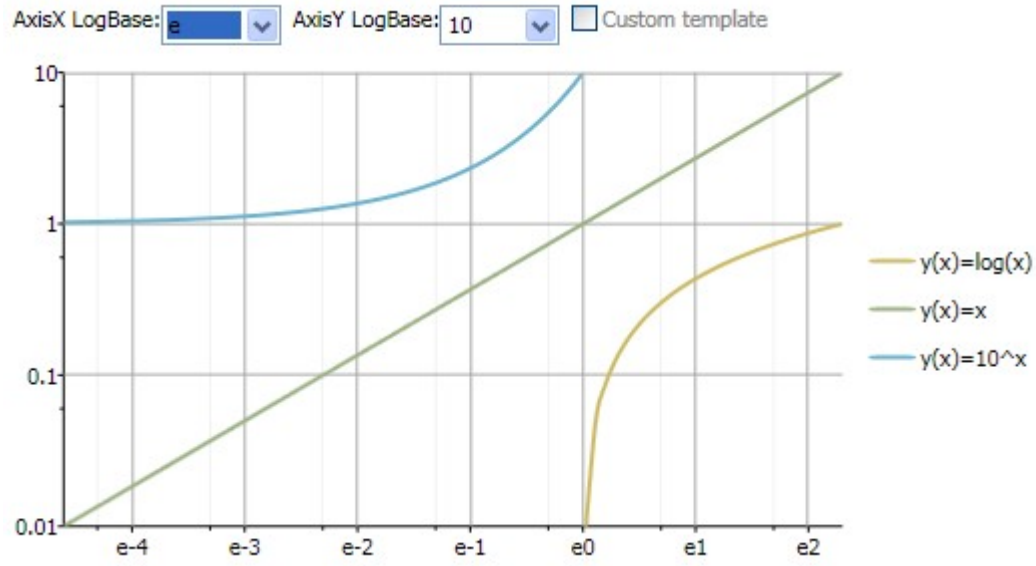
对数基数对数基数

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。

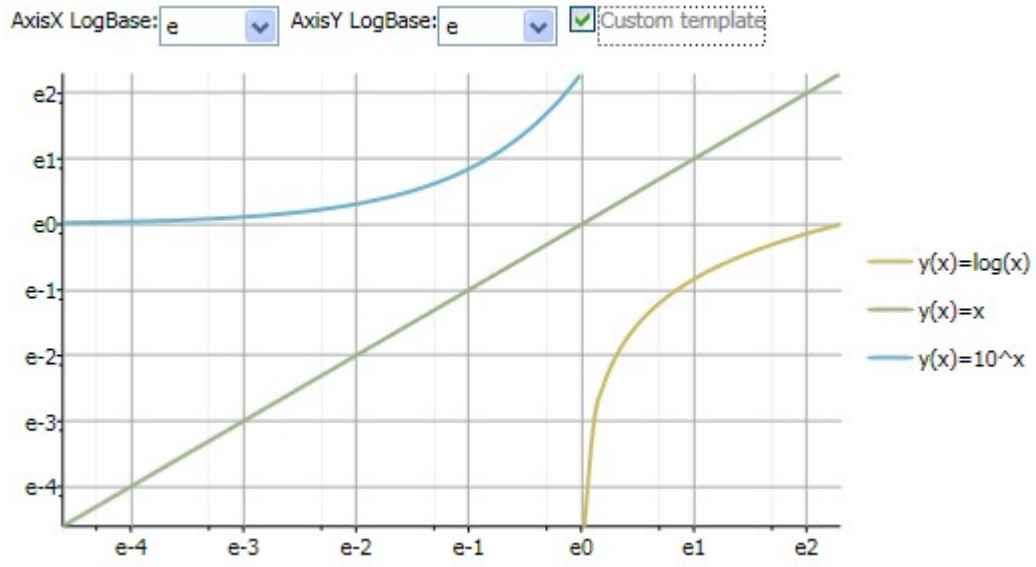
自然对数是以常数e为基数的对数。请注意，当值小于或等于零的时候，对数比例具有数学意义。
下图显示了当X-轴和Y-轴的LogBase设置为10，即常用对数时的图像：



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。