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#
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="05d97b21-befe-4052-9651-4f5273f828f8"><ac:plain-text-body><![CDATA[
c1Chart1.Reset(true);
c1Chart1.DataContext = new double[] { 1, 2, 3, 4, 5 };
c1Chart1.ChartType = ChartType.Column;
]]></ac:plain-text-b
```

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#
```

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="8a6920c3-ab03-4dfd-8b4f-5064ce314e55"><ac:plain-text-body><![CDATA[
```

```
c1Chart1.Reset(true); c1Chart1.DataContext
new Point[] { new Point(1, 1), new Point(2
2), new Point(3,
]]></ac:plain-text-body></ac:structured-ma
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"));
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="9ce4a353-9ba4-4337-a952-a1cc2405a908"><ac:plain-text-body><![CDATA[chart.Bindings = bindings;
chart.DataContext = new InOut[]
]]></ac:plain-text-body></ac:structured-macro>
{ 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#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="84bc669b-69b2-4950-b435-b35e87287de3"><ac:plain-text-body><![CDATA[
```

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

```
]]></ac:plai
```

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：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。

类类	值属性值属性	值绑定属性值绑定属性
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

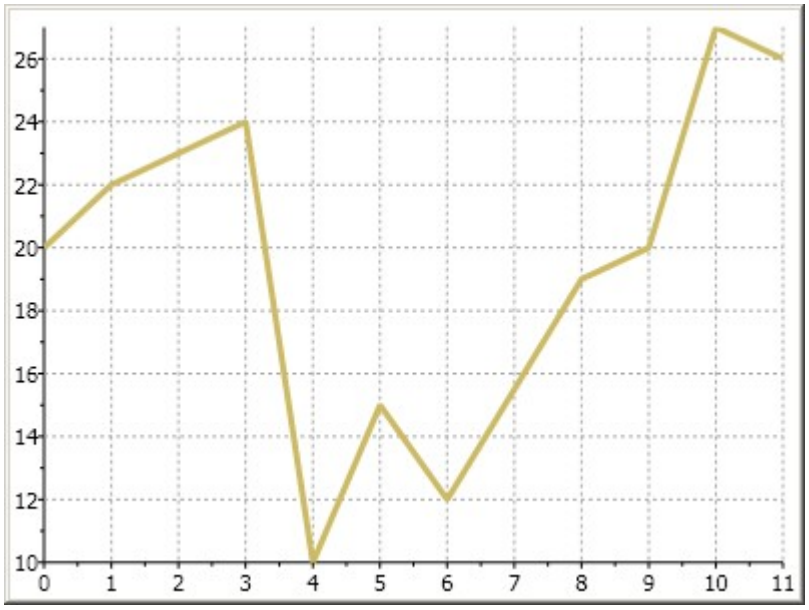
显示金融数据为一组标记最高值、最低值、开盘价以及收盘价的值的线图表数据系列外观
每个数据系列的外观由DataSeries（在线文档 'DataSeries 类'）类的三组属性决定：Symbol，Connection，以及ConnectionArea。这些属性会根据图表类型的不同影响图表的不同部分。
Symbol属性确定每个数据点绘制的符号的形状、大小、轮廓和填充属性。它们适用于图表中会显示符号类型，包括折线图、面积图、以及XYPlot图表。Symbol 属性也控制条形图和柱状图中条形的外观。
Connection
属性决定了数据点之间的线条的轮廓和填充属性。它们适用于数据系列的所有点集合。对于折线图，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

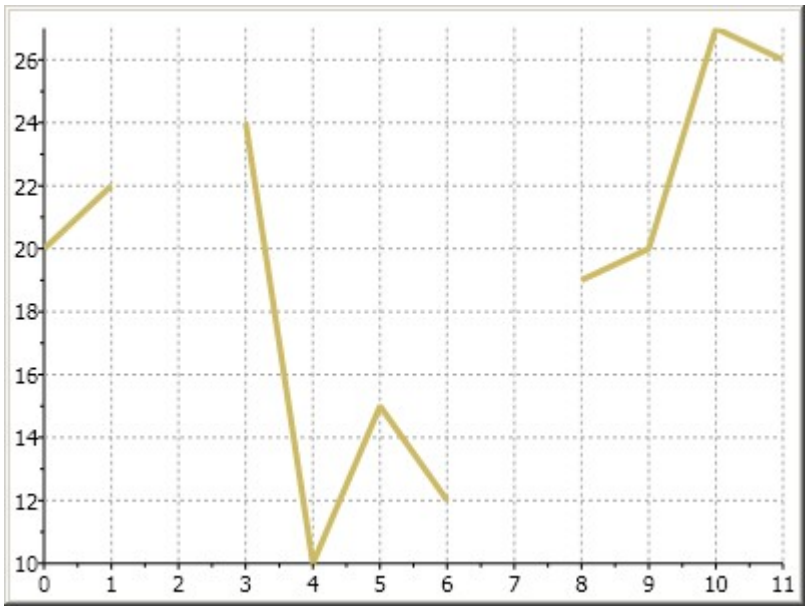
```
Me.C1Chart1.Data.Children(1).Display = C1.WPF.C1Chart.SeriesDisplay.ShowNaNGap
```

C#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="4198cc72-a0f0-4a0f-91fa-3074e50dfd9a"><ac:plain-text-body><![CDATA[
```

```
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

ClChart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。