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

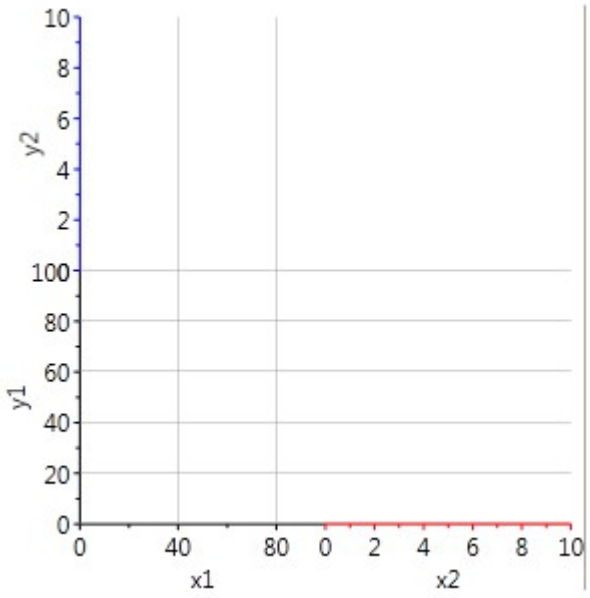
```
<!-- vertical line with coordinate label -->
<c1chart:ChartPanelObject x:Name="xmarker" DataPoint="0,NaN"
VerticalAlignment="Stretch"
Action="LeftMouseButtonDrag">
<Border BorderBrush="Red" BorderThickness="3,0,0,0"
Margin="-1.5,0,0,0" >
<TextBlock
Text="{Binding RelativeSource={RelativeSource Self},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

渲染模式渲染模式	局限性局限性

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。

1. 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(...);
}
}
```