WPF及Silverlight版Maps概述

WPF及Silverlight版Maps支持视图的平滑缩放、平移以及屏幕与地理坐标之间的映射，并将它们置于工具栏中预览。**C1Map**允许您从不同的资源中获取丰富的地理信息，如Bing Maps，Google Maps。

基于Microsoft Deep Zoom技术的**C1Maps**，可以让最终用户享受到极致的特写镜头、高清晰度的图像以及平滑的过渡。它同时也支持图层操作方便您在地图上添加自定义元素。

WPF及Silverlight版帮助文档

入门

- 关于**ComponentOne Studio WPF Edition**安装，授权，技术支持，命名空间及附有控件的工程创建，请参阅 Getting Started with WPF Edition。
- 关于**ComponentOne Studio Silverlight Edition**安装，授权，技术支持，命名空间及附有控件的工程创建，请参阅Getting Started with Silverlight Edition。

WPF及Silverlight版Maps主要特性

WPF及Silverlight版Maps可以让您创建个性化以及丰富的应用程序，通过采用以下主要特性，您可以充分的体验**WPF及Silverlight版Maps**：

- 几何图形的绘制
  C1Maps的矢量层允许您在地图上通过地理来坐标绘制几何/形状/多边形/路径，矢量层可以有效地绘制以下内容：

  - 政治边界（如国家或州）。
  - 地理信息（如显示公路或者飞机航线）。
  - 等值线图（如通过统计数据显示每个国家的人口数）。

  您可以使用矢量层来显示整个世界地图，而不是采用常规的Microsoft Virtual Earth资源。http://www.componentone.com/newimages/Products/Screenshots/StudioSilverlight/C1MapsVectorAsChart.png

- 支持KML
  矢量层支持基本的KML导入/导出（KML是地图上图形交换的标准文件格式），了解更多相关信息，请参阅**KML导入与导出**。

- 丰富的地理信息
  您可以从Bing Map或任何自定义资源中展示丰富地理信息，例如您可以在Yahoo！地图上创建您自己的资源。

- 地图上大量的元素显示
  WPF及Silverlight版Maps允许本地和服务器数据的虚拟化，使用虚拟层地图仅能显示和请求当前可见的地图元素。

- 缩放，平移及地理坐标
  WPF及Silverlight版Maps支持通过鼠标或者键盘完成缩放和平移操作，它同时也支持在屏幕和地理坐标之间的映射。

- 支持图层
  您可以通过图层将您自定义的元素添加到地图上，元素可以与地理位置连接，了解更多信息，请参阅**矢量层**，**虚拟化**及**项目分层**。

- 支持Silverlight Toolkit主题
  添加当下最流行的Microsoft Silverlight Tookit内置主题样式到您的UI中，其中包括ExpressionDark，ExpressionLight，WhistlerBlue，RainierOrange，ShinyBlue，及BureauBlack，请参阅**C1Maps主题**。

WPF及Silverlight版Maps快速入门

快速入门可以帮助您顺利的创建与运行**WPF及Silverlight版Maps**，您将在Expression Blend中创建一个附有**C1Maps**控件的新项目，当控件被添加时，您就可以自定义它的外观并为它添加**C1VectorLayer**及**C1VectorPlacemark**，以及创建一个数据源，此时您可以对**C1VectoPlacemark**属性绑定数据源。在快速入门的最后，您将拥有一个包含一系列地标标签的全功能地图控件。

第一步：创建一个附有C1 Maps控件的应用程序

在这一步中，您将使用**C1Maps**控件在Expression Blend中创建一个**WPF及Silverlight**应用程序，您也可以设置控件的属性。

完成如下步骤：

1. 在Expression Blend中，选择**文件|新建工程**。
2. 在**新建工程**对话框中，于左侧面板中选择Silverlight项目类型，在右侧面板选择**WPF应用程序+Website**或者**Silverlight应用程序+Website**。
3. 输入您项目的**名称**和**位置**，在下拉框中选择**语言**并点击**确定**，此时Blend创建了一个新应用并在设计面板中显示**MainPage.xaml**文件。
4. 通过完成以下步骤，为**C1.WPF.Maps**或**C1.Silverlight**添加一个引用：
   a. 选择**项目| 添加引用**。
   b. 浏览并查找**C1.WPF.Maps.dll**或**C1.Silverlight.Maps.5.dll**为**WPF版Maps**进行配置安装。

   > **注意：** 所有**.dll**文件默认安装在**C:\Program Files\ComponentOne\WPF Edition**和**C:\Program Files\ComponentOne\Silverlight Edition**目录下。

   c. 选择**C1.WPF.dll**并点击**打开**，此时您的项目即添加了一个引用。
5. 完成以下步骤为您的项目添加**C1Maps**控件：
   a. 在菜单栏选择**窗口|资源**，打开**资源**标签。
   b. **资源**标签下，在查找框中输入“C1Maps”。
   c. 此时出现**C1Maps**控件图标。
   d. 双击**C1Maps**图标将控件添加到您的项目中。
6. 在**对象和时间线**面板中选择**C1Maps**，此时对**属性**面板中的属性设置如下：
   - 设置**Name**属性为"C1Maps1"，使您的控件有唯一的标识来调用。
   - 设置**Width**属性为"405"。
   - 设置**Height**属性为"472"。
   - 设置**C1Maps.Zoom**属性为"2"， 此时缩放比例是原始的2倍。
   - 设置**C1Maps.Center**属性为"-65, -25"，此时只有南美洲出现在地图上。

在这一步中，您创建了一个Blend工程并且添加了一个**C1Maps**控件，此外您还为**C1Maps**控件设置了属性。

第二步：绑定数据源

在这一步中您将创建一个类，类中包含**Name**和**LongLat**两个属性并封装到数组集合，此外您还需要为控件添加一个包含**C1VectorPlacemark**的**C1VectorLayer**，此时**Name**将与C1VectorPlacemark的标签属性绑定，**LongLat**与C1VectorPlacemark的地理坐标属性绑定。

完成如下步骤：

1. 打开**MainPage.xaml**代码页面 （根据您所选择项目的语言，可能是MainPage.xaml.cs或MainPage.xaml.vb）。
2. 将下述的类添加到您的项目中，需要将它置于命名空间声明的下方。
3. 这个类生成的对象包含两种属性：一个字符串类型的**Name**及一个**Point**类型的**LongLat**。

Visual Basic

```
Public Class City
Private _LongLat As Point
    Public Property LongLat() As Point
        Get
            Return _LongLat
        End Get
        Set(ByVal value As Point)
            _LongLat = value
        End Set
    End Property

Private _Name As String
    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property

    Public Sub New(ByVal location As Point, ByVal cityName As String)
        Me.LongLat = location
        Me.Name = cityName
    End Sub
End Class
```

C#

```
public class City
  {
    public Point LongLat { get; set; }
    public string Name { get; set; }
  }
```

4. 在**InitializeComponent()**方法下添加下述代码来创建包含**Name**和**LongLat**属性的数组集合：

Visual Basic

```
Dim cities() As City =
New City() {
New City(New Point(-58.40, -34.36),  "Buenos Aires"),
New City(New Point(-47.92, -15.78), "Brasilia"),
New City(New Point(-70.39, -33.26), "Santiago"),
New City(New Point(-78.35, -0.15), "Quito"),
New City(New Point(-66.55, 10.30), "Caracas"),
```

```
New City(New Point(-77.03, -12.03), "Lima"),
New City(New Point(-57.40, -25.16), "Asuncion"),
New City(New Point(-74.05, 4.36), "Bogota"),
New City(New Point(-68.09, -16.30), "La Paz"),
New City(New Point(-58.10, 6.48), "Georgetown"),
New City(New Point(-55.10, 5.50), "Paramaribo"),
New City(New Point(-56.11, -34.53),"Montevideo")
}
C1Maps.DataContext = cities
```

C#

```
City[] cities = new City[]
{
   new City(){ LongLat= new Point(-58.40, -34.36), Name="Buenos Aires"},
   new City(){ LongLat= new Point(-47.92, -15.78), Name="Brasilia"},
   new City(){ LongLat= new Point(-70.39, -33.26), Name="Santiago"},
   new City(){ LongLat= new Point(-78.35, -0.15), Name="Quito"},
   new City(){ LongLat= new Point(-66.55, 10.30), Name="Caracas"},
   new City(){ LongLat= new Point(-56.11, -34.53), Name="Montevideo"},
   new City(){ LongLat= new Point(-77.03, -12.03), Name="Lima"},
   new City(){ LongLat= new Point(-57.40, -25.16), Name="Asuncion"},
   new City(){ LongLat= new Point(-74.05, 4.36), Name="Bogota"},
   new City(){ LongLat= new Point(-68.09, -16.30), Name="La Paz"},
   new City(){ LongLat= new Point(-58.10, 6.48), Name="Georgetown"},
   new City(){ LongLat= new Point(-55.10, 5.50), Name="Paramaribo"},
};
C1Maps.DataContext = cities;
```

5. 切换到XAML页面并修改*<c1:C1Maps>*标签，起始与终止标签如下所述：

XAML

```
<c1:C1Maps x:Name="C1Maps1" FadeInTiles="False" Margin="0,0,235,8"
TargetCenter="-65,-25" Center="-58,-25" Zoom="2">
</c1>
```

6. 将*Foreground="Aqua"*添加到*<c1:C1Maps>*标签。
7. 将下述XAML标签置于*<c1:C1Maps>*与*</c1:C1Maps>*标签之间：

XAML

```
<c1:C1Maps.Resources>
  <!--Item template -->
     <DataTemplate x:Key="templPts">
       <c1:C1VectorPlacemark
         GeoPoint="{Binding Path=LongLat}" Fill="Aqua" Stroke="Aqua"
         Label="{Binding Path=Name}" LabelPosition="Top" >
         <c1:C1VectorPlacemark.Geometry>
```

```
            <EllipseGeometry RadiusX="2" RadiusY="2" />
          </c1:C1VectorPlacemark.Geometry>
        </c1:C1VectorPlacemark>
      </DataTemplate>
  </c1:C1Maps.Resources>
  <c1:C1VectorLayer ItemsSource="{Binding}"
ItemTemplate="{StaticResource templPts}" HorizontalAlignment="Right" Width="403"
/>
```

此XAML创建了一个数据模板，一个**C1VectorPlacemark**及一个**C1VectorLayer**，**C1VectorLayer**的**ItemsSource**属性与完整的数据源绑定，当**Laber**属性被设置为**Name**属性的值时，**C1VectorPlacemark**的**GeoPoint**属性与**LongLat**属性值相绑定。当您运行项目时，**Label**和**Name**属性将封装并创建一系列地图上的地理标记。

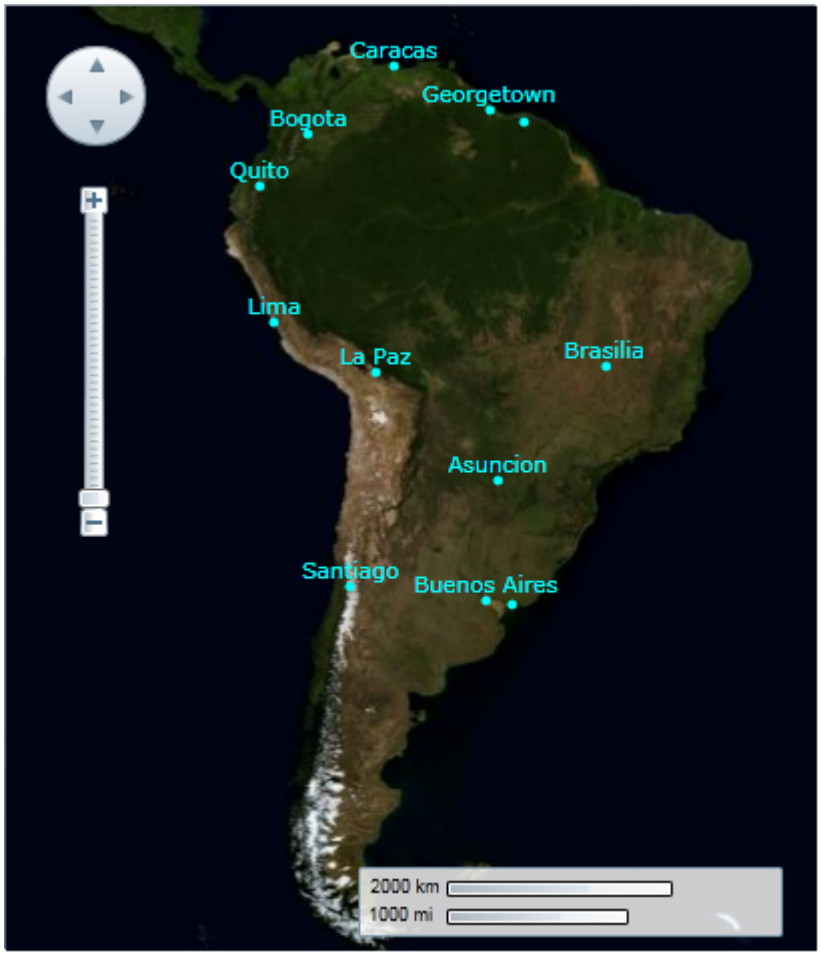
在这一步中，您创建了一个数据源并将它与C1VectorPlacemark的属性相绑定，在下一步中，您将运行程序并预览其结果。

第三步：运行项目

在上一步中，您创建了一个附有**C1Maps**控件的Blend项目，生成了一个数据源，为**C1Maps**控件添加了一个**C1VectorLayer**和一个**C1VectorPlacemark**，及为**C1VectorPlacemark**属性绑定数据源。

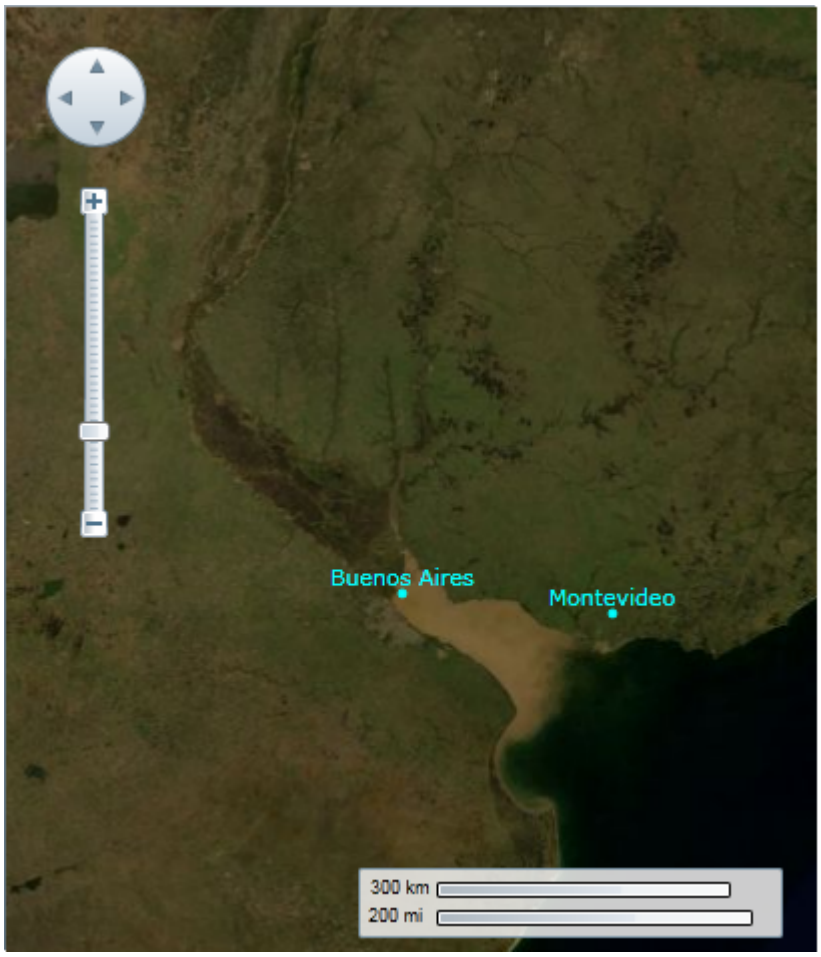
完成以下步骤：

1. 按**F5**运行项目，可以看到**C1Maps**控件显示如下：



可以观察到有两个没有命名的点，一个在**Buenos Aires**附近，另一个在**Georgetown**附近。

2. 双击**Buenos Aires**区域，重复两次此步骤，此时可以观察到一个名为**Montevideo**的标记会出现在地图上。



恭喜您！您已经完成了**WPF及Silverlight版Maps**的快速入门，我们建议您访问**C1Maps控件基础**和**WPF及Silverlight版Maps任务帮助**来熟悉更多的内容。

XAML快速参考

本主题主要提供了一个使用XAML以完成不同任务的快速概要，了解更多信息，请参阅**WPF及Silverlight版Maps任务帮助**。

项目模板

下述例子说明了如何使用项目模板

XAML

```
<c1: x:Name="C1Maps1" FadeInTiles="False" Margin="0,0,235,8" TargetCenter="-65,-25"
Center="-58,-25" Zoom="2" Foreground="Aqua">
  <c1:C1Maps.Resources>
  <!--Item template -->
     <DataTemplate x:Key="templPts">
       <c1:C1VectorPlacemark
         GeoPoint="{Binding Path=LongLat}" Fill="Aqua" Stroke="Aqua"
         Label="{Binding Path=Name}" LabelPosition="Top" >
         <c1:C1VectorPlacemark.Geometry>
           <EllipseGeometry RadiusX="2" RadiusY="2" />
         </c1:C1VectorPlacemark.Geometry>
       </c1:C1VectorPlacemark>
     </DataTemplate>
  </c1:C1Maps.Resources>
  <c1:C1VectorLayer ItemsSource="{Binding}"
ItemTemplate="{StaticResource templPts}" HorizontalAlignment="Right" Width="403" />
</c1>
```

矢量层标记

下述例子说明了如何使用矢量层创建标记:

XAML

```
<c1:C1Maps>
<c1:C1VectorLayer>
<c1:C1VectorPlacemark LabelPosition="Left" GeoPoint="-80.107008,42.16389"
StrokeThickness="2" Foreground="#FFEB1212" PinPoint="-80.010866,42.156831"
Label="Erie, PA"/>
</c1:C1VectorLayer>
```

矢量层—折线

下述例子说明了如何使用矢量层创建一个折线（开线）:

XAML

```
<c1:C1Maps>
<c1:C1VectorLayer Margin="2,0,-2,0">
     <c1:C1VectorPolyline Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
     </c1:C1VectorPolyline>
</c1:C1VectorLayer>
```

```
</c1:C1Maps>
```

矢量层—多边形

下述例子说明了如何使用矢量层创建一个折线（用折线创建一个图形）：

XAML

```
<c1:C1Maps>
<c1:C1VectorLayer Margin="2,0,-2,0">
     <c1:C1VectorPolygon Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
     </c1:C1VectorPolygon>
</c1:C1VectorLayer>
</c1:C1Maps>
```

主题化

下述例子说明了如何在**C1Maps**控件上应用一个主题：

XAML

```
<my:C1ThemeRainierOrange>
    <c1:C1Maps Height="172" Width="288" Margin="200,0,34,0"/>
</my:C1ThemeRainierOrange>
```

C1Maps控件基础

C1.WPF.Maps和**C1.Silverlight.Maps**都包含C1Maps控件，通过**C1Maps**可以从不同的资源中显示丰富的地理信息，这些资源包括Bing Maps及您的自定义数据。

C1Maps支持缩放，平移以及屏幕同地理坐标之间的映射，它同时也支持图层操作方便您在地图上叠加您需要的元素，图层操作支持项目虚拟化并可以显示如地理坐标这样的静态元素。

以下主题将介绍**C1Maps**控件的基础。

法律要求

C1Maps允许您使用**Bing Maps**的相关地理信息，在使用此项服务前，您应该检查该服务相关的许可要求，这些许可条款您可以在这里找到：

http://www.microsoft.com/maps/product/terms.html

HTTPS支持

Microsoft Silverlight出于安全性的考虑限制了跨区域，跨领域及跨方案的URL访问，以下表格总结了相关规则：

	下载对象	媒体，图像，ASX	XAML文件, Font文件	流媒体
允许的方案	HTTP, HTTPS	HTTP, HTTPS, FILE	HTTP, HTTPS, FILE	HTTP
跨方案访问	否	否	否	HTTPS以外
跨Web领域访问	否	HTTPS以外	否	是
跨区域访问(Windows)	否	否	否	否
跨区域访问(Macintosh)	否	是	否	是
允许重定向	相同领域(仅限Firefox/Safari)	相同领域	相同领域	否

了解Silverlight HTTPS支持的更详细信息，请在MSDN中参阅**Silverlight URL访问策略**：http://msdn.microsoft.com/en-us/library/bb820909.aspx.

> **注意：C1**控件可以使用HTTPS协议，然而图像图块必须源自Silverlight应用的相同领域。

C1Maps概念和主要属性

本主题将详述C1Maps的基本概念及主要属性的描述。

地图资源

C1Maps可以从不同的资源中显示地理信息，在默认情况下，**C1Maps**使用**Microsoft LiveMaps**航拍照片作为资源，但您也可以通过**MultiScaleTileSource**对象改变**Source**属性。

您需要加入下列代码：

- **虚拟地球航拍资源**

Visual Basic

```
map1.Source = new VirtualEarthAerialSource()
```

C#

```
map1.Source = new VirtualEarthAerialSource();
```

- **虚拟地球路拍资源**

Visual Basic

```
map2.Source = new VirtualEarthRoadSource()
```

C#

```
map2.Source = new VirtualEarthRoadSource();
```

- **虚拟地球混合资源**

Visual Basic

```
map3.Source = new VirtualEarthHybridSource()
```

C#

```
map3.Source = new VirtualEarthHybridSource();
```

可视化地图

地图上当前的可见部分是由**Center**和**Zoom**属性及控件的尺寸来决定的：

Center属性类型为**Point**，其中X属性表示地理坐标的经度，Y属性表示纬度，用户可以在地图上用拖拽鼠标来改变**Center**属性的值，或者采用左上角的导航控制。

Zoom属性指定了地图当前的缩放率，当缩放值为0时地图为完整的缩小地图，缩放值每增加1，地图的分辨率就增大一倍，用户可以通过鼠标的滑轮或者左上角的缩放控制改变**Zoom**的属性。

C1Maps采用的三种坐标系：

- **地理**坐标系采用经度和纬度来定位地点，这种坐标系未采用笛卡尔坐标，因此当您平移时地图的比例可能会改变。
- **逻辑**坐标系在整个地图上的坐标轴的取值范围从0到1，由于这种坐标系属于笛卡尔坐标，因此更方便用户来使用。
- **屏幕**坐标系是相对于左上角控件的像素坐标，这种坐标系在定位控件和处理鼠标事件上，更容易被使用。

C1Maps为这些坐标系之间转换提供了四种方法：

ScreenToGeographic，**ScreenToLogic**，**GeographicToScreen**及**LogicToScreen**，地理坐标系与逻辑坐标系之间的转化可以通过使用**C1Maps.Projection**属性完成投影配置，投影可以被更改以支持不同的地图，默认情况下采用**LiveMaps**及多数其他供应商所使用的墨卡托投影。

信息层

此外在地图源所提供的地理信息上，您还可以为地图添加信息层，C1Maps默认包含了五种：

- **C1MapItemsLayer**可以显示地图上任意的地理位置项，该层是一个**ItemsControl**。因此它支持直接添加**UIElement**对象或者包含DataTemplate的通用数据对象，该DataTemplate可以转换为虚拟项。
- **C1MapVirtualLayer**可以显示虚拟项，这意味着只有当地图上的区域是可见时它才会被载入，同时它也支持异步请求，只有当他们出现在视图中时，新的项才会从服务器中被加载。

- **C1VectorLayer**可以显示矢量数据，如线以及地理位置定位点组成的多边形，同时他也可以从KML文件中保存和加载。
- **C1MapToolsLayer**是下一层，可以显示包括平移，缩放及比例的工具栏，该层被用来构建**C1Maps**的模板，因此没有必要对其手动进行添加。
- **C1MapTilesLayer**是地图图块显示的背景层，您通常不必使用该层，因为它由C1Maps自动管理。

> **注意：C1Maps**仅适用于包含Web站点或Web应用的项目，如果您仅在一个单独的Silverlight解决方案中使用，它将不会显示任何内容。

项目分层

C1MapItemsLayer可以用最简单的方式显示地图上的项，它继承了**ItemsControl**，因此它支持直接添加**UIElement**对象或包含**DataTemplate**的通用数据对象，该**DataTemplate**可以转换为虚拟项，我们可以通过**C1MapCanvas.LatLong**为**C1MapItemsLayer**添加元素属性，请看下面一个例子：

XAML

```
<c1:C1Maps>
    <c1:C1Maps.Layers>
        <c1:C1MapItemsLayer>
            <Ellipse Width="20" Height="20" Fill="Red"
                     c1:C1MapCanvas.LatLong="-79.9247, 40.4587"
                     c1:C1MapCanvas.Pinpoint="10, 10"/>
        </c1:C1MapItemsLayer>
    </c1:C1Maps.Layers>
</c1:C1Maps>
```

该示例在XAML中创建了一个**C1Maps**控件并在它的**Layers**集合中添加了一个**C1MapItemsLayer**，**Layer**集合中可以添加任意数量的层，多个层以叠加方式显示。

我们将在项目层中添加一个纬度/经度为（40.4587，-79.9247）椭圆形项，注意到在XAML中经纬度数据顺序是相反的，这是因为**LatLong**的类型**Point**结构中，X值代表的是经度，Y值代表的是纬度（这种匹配方式与地图上通常使用的X/Y轴相似）。

在上述的示例中，可以看到我们使用了**C1MapCanvas.Pinpoint**属性，该属性配置了在元素内部与**LatLong**属性中的地理坐标相匹配的点，在示例中**Pinpoint**被设置为（10,10），因此椭圆将位于**LatLong**的中心。

接下来让我看第二个示例，此时我们将用代码创建一个**C1Maps**控件并封装数据，我们将使用下述类：

C#

```
public class Place
 {
     public string Name { get; set; }
     public Point LatLong { get; set; }
 }
And here is the example code:
var map = new C1Maps();
 var itemsLayer = new C1MapItemsLayer
 {
     ItemsSource = new[]
     {
```

```
        new Place {
            Name = "ComponentOne",
            LatLong = new Point(-79.92476,  40.45873), },
        new Place {
            Name = "Greenwich Park",
            LatLong = new Point(  0.00057,  51.47617), },
    },
    ItemTemplate = itemTemplate
};
map.Layers.Add(itemsLayer);
```

我们在**Place**类的一个实例中封装了**ItemsSource**，并且我们在下述页面资源中定义的**DataTemplate**设置为**ItemTemplate**：

XAML

```
<DataTemplate x:Key="itemTemplate">
    <StackPanel Orientation="Horizontal"
                c1:C1MapCanvas.LatLong="{Binding LatLong}"
                c1:C1MapCanvas.Pinpoint="5, 5">
        <Ellipse Fill="Red" Width="10" Height="10" />
        <TextBlock Text="{Binding Name}" Foreground="White" />
    </StackPanel>
</DataTemplate>
```

该**DataTemplate**将**C1MapCanvas.LatLong**绑定到项目中定义的**LatLong**，并且在**TextBlock**中显示地域名称。

使用**ItemTemplate**和**ItemsSource**可以很方便的加载数据库中的数据，您只需要安装一个Web服务器来返回一个数据对象集，设置集合为**ItemsSource**，并为**DataTemplate**绑定一个合适的值。

虚拟化

C1MapVIrtualLayer可以实现地图虚拟化和地图元素显示的数据异步载入，因此只要不在同一时间可见，它就可以显示无穷多的元素。它的对象模型完全不同于**C1MapItemsLayer**，**C1MapVirtualLayer**需要对地图上的区域进行划分，此外，项目的资源必须在**IMapVirtualSource**接口中实现。

我们采用**MapSlice**集合中的**C1MapVirtualLayer.Slices**完成对地图上的区域划分，每个地图片定义为最小缩放级，该片的最大缩放级是下一片的最小缩放级（或者，如果他是最后一片，那它的最大缩放级就是地图最大的缩放级），每一片依次按照纬/经度的网格进行划分。

参考以下图层示例:

C#

```
var layer = new C1MapVirtualLayer
{
    Slices =
    {
        new MapSlice(2, 2, 5),
        new MapSlice(4, 4, 10)
    }
};
```

例如有两个地图片，一个缩放范围从5到10，另一个从10到最大缩放值，当缩放值从一个片到另一个片，虚拟层会从它的资源取得数据，假如第一片会得到一个2*2的经纬度区域，这就意味着该片被划分为四个区域，并且该层仅能从当前可见的区域中请求数据，第二个片可以被划为16个区域，需要更高的缩放值则必须更多的划分。

为了更好的理解**IMAPVirtualSource**接口，让我们来看一个工厂方法的实现示例：

C#

```
public class ServerStoreSource : IMapVirtualSource
{
    public void Request(double minZoom, double maxZoom,
                        Point lowerLeft, Point upperRight,
                        Action<ICollection> callback)
    {
        if (minZoom < minStoreZoom)
            return;
        var client = CreateFactoriesService();
        client.GetStoresCompleted += (s, e) =>
        {
            if(e.Error == null)
                callback(e.Result);
        };
        client.GetStoresAsync(lowerLeft.Y, lowerLeft.X,
                              upperRight.Y, upperRight.X);
    }
}
```

Request方法将地图上的一个区域作为参数，并返回得到一个项目集合，这种特殊的实现首先需要检查最小的缩放请求是否小于应用的最小值参数，如果为真则什么都不做，否则就触发Web服务去获取数据。

服务器端通过**GetStores**方法实现，它遍历数据库中的所有元素，并返回该请求范围内的项目：

C#

```
public List<Store> GetStores(double lowerLeftLat, double lowerLeftLong,
    double upperRightLat, double upperRightLong)
{
    var stores = new List<Store>();
    var dataBase = DataBase.GetInstance(Context);
    foreach (var store in dataBase.Stores)
    {
        if (store.Latitude > lowerLeftLat
          && store.Longitude > lowerLeftLong
          && store.Latitude <= upperRightLat
          && store.Longitude <= upperRightLong)
        {
            stores.Add(store);
        }
    }
    return stores;
}
```

更好的一种实现方法是将已经划分好的区域保存起来，以防止他们被遍历。

矢量层

矢量层允许您将不同的地理坐标对象置于地图上。

矢量对象

下述为矢量层上主要的矢量对象：

- **C1VectorPolyline**—这个对象除了不需要封闭图形，其他与**Polygon**类相类似，折线以地理坐标构成，典型的应用包括：路径和路由。更多的相关任务的帮助，请查阅**添加一个折线**。
- **C1VectorPolygon**—与**Polyline**类相类似，但需要多画一条折线用以将一系列直线构成一个封闭的图形。多边形是由多个地理坐标点构成的，典型的应用包括：边界和区域。更多的相关任务的帮助，请查阅**添加一个多边形**。
- **C1VectorPlacemark**—将一个对象添加到地理位置上，这个地理标志拥有独特的几何形状，它的坐标可以由像素坐标或者选择标签（任意的**UIElement**）表示，典型的应用包括：标志，图标及地图上的标记。更多相关任务的帮助，请参阅**添加一个标签**。

元素可见性

根据当前地图的比例，可以通过几个属性来控制元素的可见，例如，当地图放大时您可以看到内部的细节，当地图被缩小时将会隐藏这些细节。

C1VectorLayer.MinSize属性可以实现全局控制，通过指定最小线性屏幕尺寸来确定元素变为可见的界限。

通过一个特殊的属性C1VectorPlacemarklabels.C1VectorLayer.LabelVisibilty来控制可见性，它可以使用以下值：

- **Hide—**标签不可见，仅在工具提示中显示。
- **AutoHide—**重叠时隐藏。
- **Visible—**所有标签可见。

此外，每一个矢量元素都有自己的可见性设置，它们被保存在LOD属性中并高于全局变量优先级。

LOD （细节等级）结构具有如下属性：

- **MinSize**, **MaxSize—**指定在可见范围内元素的线性屏幕尺寸大小，如果大小不在元素尺寸的范围内则会被隐藏。
- **MinZoom**, **MaxZoom—**另外您可以指定显示元素地图的比例（C1.Zoom属性）。

KML导入与导出

KML是一种基于XML的语言，它主要解决地理信息和注释的可视化的问题，最早在Google Earth上广泛使用，了解更多信息请参阅https://developers.google.com/kml/documentation.

KML通过**KMLReader**类实现导入的过程，它包含一个静态的方法可以从KML资源（串或者流）中创建一个矢量对象集，您可以很轻易将该集加入到**C1VectorLayer**。与此同时，导入的**DataContext**对象对应KML中的**XElement**，因此您可以在导入过程中使用原始元素执行自定义操作。

导入限制：

- 仅支持KML位置标记元素。
- 不支持内部多边形。
- 不支持图标。
- 不支持外部链接。

KML通过**KMLWriter**类实现导出的过程，它包含一个静态方法可以将一个矢量对象集转换为KML资源。

KMLWriter.Write()方法中的saveElementCallback属性允许您在导出过程中执行自定义操作，调用该方法可以将每个元素保存到KML流中，例如，您可以调用callback方法为元素添加KML自定义数据。

导出限制：

- **C1VectorPlacemark.Geometry**属性不能在KML流中保存。

数据绑定

C1VectorLayer 包含两个支持数据绑定的属性：

- **ItemsSource** — 指定源对象集合。
- **ItemTemplate —** 指定图层上每个对象的外观，同时需要在项目模板中定义一个类，该类需继承**C1VectorItemBased**。

数据绑定示例

假设您有一个**城市**对象集：

C#

```
public class City
  {
    public Point LongLat { get; set; }
    public string Name { get; set; }
  }
```

模板定义了如何从**城市**类中创建**C1VectorPlacemark**。

XAML

```
<c1:C1Maps x:Name="maps" Foreground="LightGreen">
  <c1:C1Maps.Resources>
  <!-- Item template -->
     <DataTemplate x:Key="templPts">
       <c1:C1VectorPlacemark
         GeoPoint="{Binding Path=LongLat}" Fill="LightGreen" Stroke="DarkGreen"
         Label="{Binding Path=Name}" LabelPosition="Top" >
         <c1:C1VectorPlacemark.Geometry>
           <EllipseGeometry RadiusX="2" RadiusY="2" />
         </c1:C1VectorPlacemark.Geometry>
       </c1:C1VectorPlacemark>
     </DataTemplate>
  </c1:C1Maps.Resources>
  <c1:C1VectorLayer ItemsSource="{Binding}"
  ItemTemplate="{StaticResource templPts}" />
</c1:C1Maps>
```

最后，您需要一些实例作为数据源。

C#

```
City[] cities = new City[]
{
    new City(){ LongLat= new Point(30.32,59.93), Name="Saint Petersburg"},
    new City(){ LongLat= new Point(24.94,60.17), Name="Helsinki"},
    new City(){ LongLat= new Point(18.07,59.33), Name="Stockholm"},
    new City(){ LongLat= new Point(10.75,59.91), Name="Oslo"},
    new City(){ LongLat= new Point(12.58,55.67), Name="Copenhagen"}
};

maps.DataContext = cities;
```

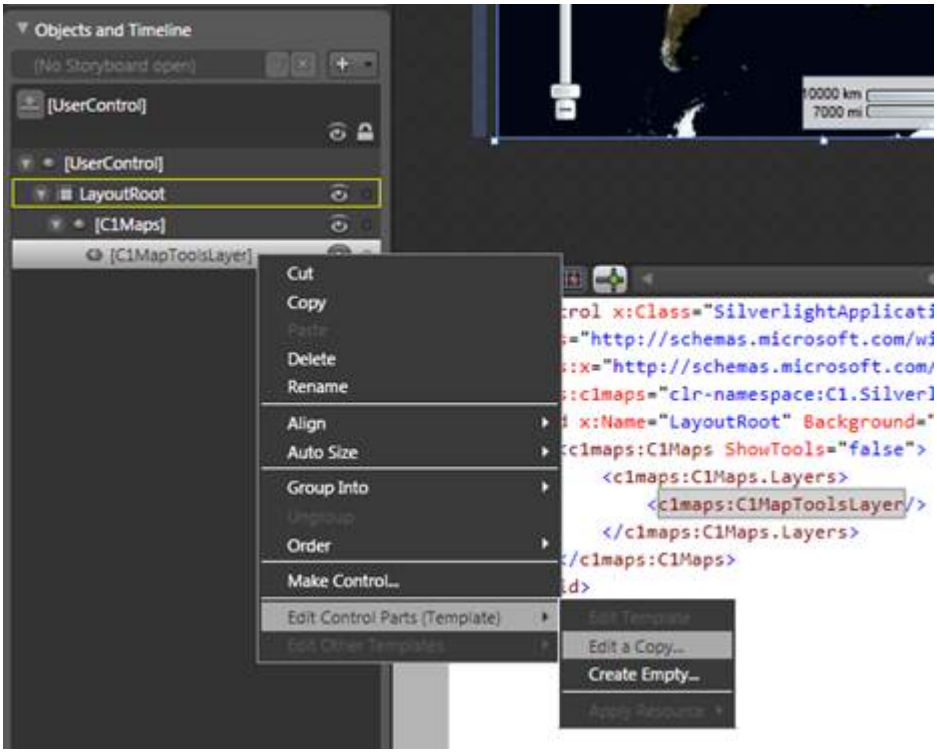
自定义工具

平移及缩放工具默认在地图中显示，并通过**C1MapToolsLayer**实现，然而它被集成在**C1Maps**的模板中，所以我们不需要将它加入到图层集合中。为了实现工具的自定义，您需要先把**C1Maps.ShowTools**置为**False**，此时工具被设置为隐藏，此后再为**C1MapToolsLayer**添加实例，以下是相关的XAML：

XAML

```
<c1:C1Maps ShowTools="false">
    <c1:C1Maps.Layers>
        <c1:C1MapToolsLayer/>
    </c1:C1Maps.Layers>
</c1:C1Maps>
```

注意，此时您需要实现一个与之前不同的工具层，但仅需对示例中内置的工具模板进行修改，就可以实现您的自定义工具。首先，您需要在Blend中编辑XAML，并右键点击**ToolsLayer**，选择**编辑控件（模板）|编辑拷贝**：



现在您就可以在Blend中编辑您的模板，且该模板的变化可以在地图中得以展现。

WPF及Silverlight版Maps的布局和外观

下述主题将详细介绍如何自定义**C1Maps**控件的布局和外观，您可以使用内置布局选项对您面板中的控件进行布局，如网格或者画布。主题还允许您自定义网格的外观，并且可以利用基于XAML的Silverlight样式。此外，您可以采用模板对控件的布局和格式进行统一，并自定义控件的行为。

如何使用ClearStyle

控件样式的每个关键部分都应该是可以直接操作的，如一个简单的颜色属性，这就导致需要对每个控件制定一系列独特的样式属性。例如，一个Gauge有PointerFill和PointerStroke两种属性，而一个DataGrid具有SelectedBrush和MouseOverBrush两种行特性。

比方说，在您的窗体里有一个控件不支持ClearStyle，您可以通过ClearStyle创建XAML资源并以此匹配您窗体中的其他其他控件（如抓取确切的颜色），或者您想要利用ClearStyle覆盖样式的部分内容（如自定义的滚动条），这都可以通过ClearStyle实现，因为ClearStyle是可扩展并可以覆盖您需要的样式区域。

ClearStyle旨在成为一个快速、便捷的样式改变解决方案，但您依然可以采取老式的ComponentOne控件样式获取需方法，而且ClearStyle不会干预那些不常用的属性，那些属性通常需要完整的自定义设计。

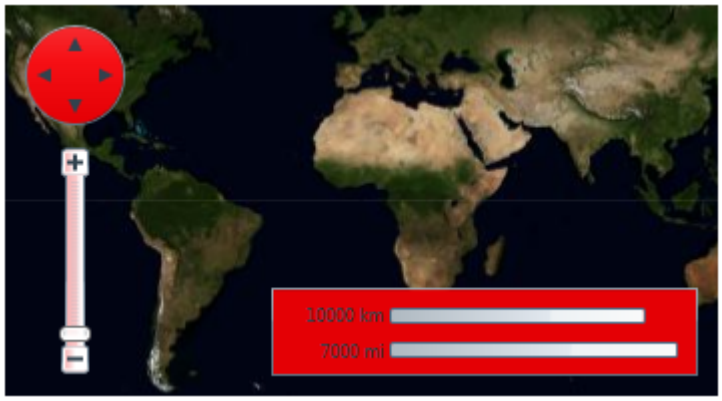
C1Maps ClearStyle属性

WPF及Silverlight版Maps支持ComponentOne's ClearStyle技术，可以帮助您不用改变控件的模板就可以轻松的对控件的颜色做出更改。通过设置几个颜色的属性，您就可以快速的改变整个网格的样式。

下表描述了**C1Maps**控件中画笔的属性：

属性	描述
Background	获取或设置控件背景的颜色。
MouseOverBrush	获取或设置**System.Windows.Media.Brush**，用以控制鼠标滑过地图上按钮时高亮的颜色。
PressedBrush	获取或设置**System.Windows.Media.Brush**，用以控制当按钮被点击时高亮的颜色。

您可以通过一系列属性的设置改变**C1Maps**的外观，例如可以通过**Background**属性改变地图工具的背景颜色，当**Background**属性设置为“##FFE40005”时，**C1Maps**控件的外观将与下图相似：

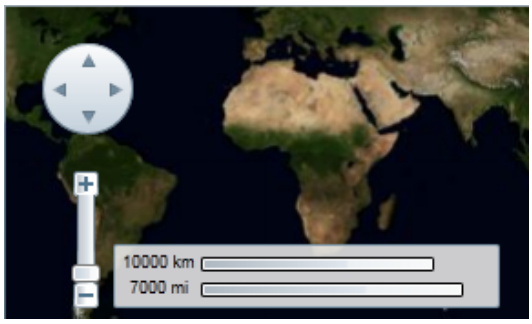


以上是ComponentOne's ClearStyle的一个简单示例，更多ClearStyle相关信息，请参阅ComponentOne ClearStyle Technology专题。

C1Maps主题

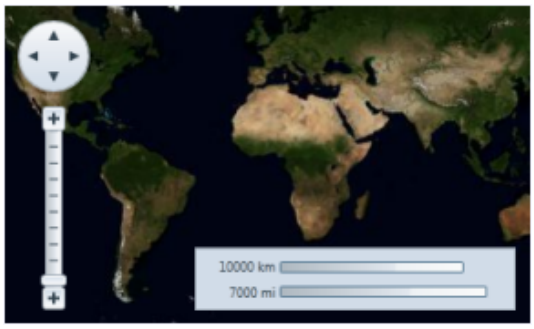
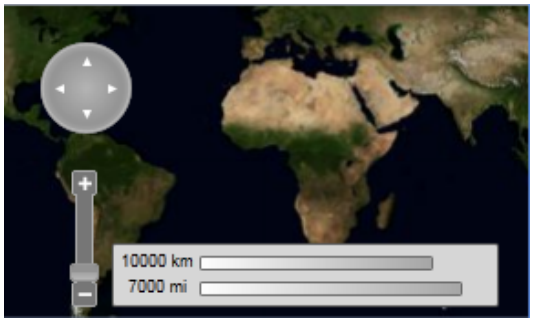
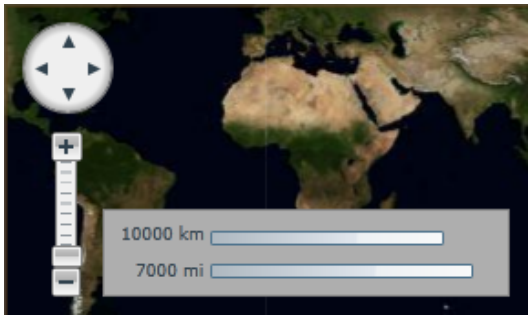
Silverlight主题是一组图像设置的集合，它定义了一个或者多个控件的外观，使用主题的优势在于您可以同时为程序中多个控件应用主题，因此它提供了一致且无重复的样式集合。

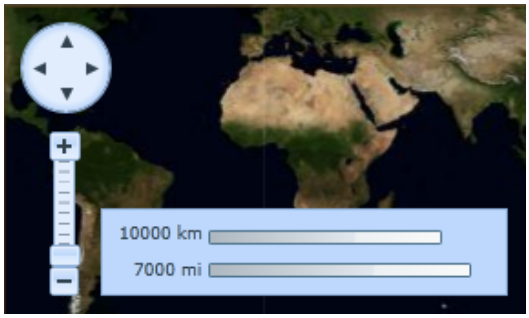
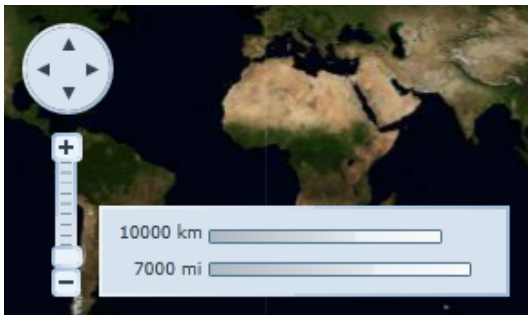
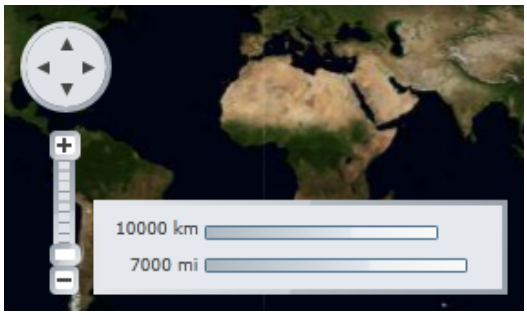
当您在项目添加了C1Maps控件时，它会出现如下图的默认主题：

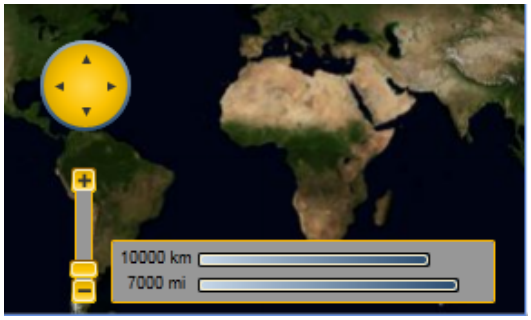


您可以通过使用内置主题或者您自定义的主题来改变外观，所有内置主题均基于WPF Tookit主题包，其描述和外观如下表：

Full Theme Name	Appearance
C1ThemeBureauBlack	

C1Blue	
C1ThemeCosmopolitan	
C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1ThemeOffice2007Black	

C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	

C1ThemeRainierOrange	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

我们通过调用**ApplyTheme**方法设置一个主题元素，首先为主题添加一个引用来配置您的项目，之后再为主题添加下述代码：

Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' 调用ApplyTheme方法
   C1Theme.ApplyTheme(LayoutRoot, theme)
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //调用ApplyTheme方法
      C1Theme.ApplyTheme(LayoutRoot, theme);
   }
```

你需要使用System.Windows.ResourceDictionary.MergedDictionarie属性将主题应用于整个应用程序，首先为主题添加一个引用来配置您的项目，之后再为主题添加下述代码：

Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' 调用MergedDictionaries方法
```

```
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //调用MergedDictionaries方法
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
   }
```

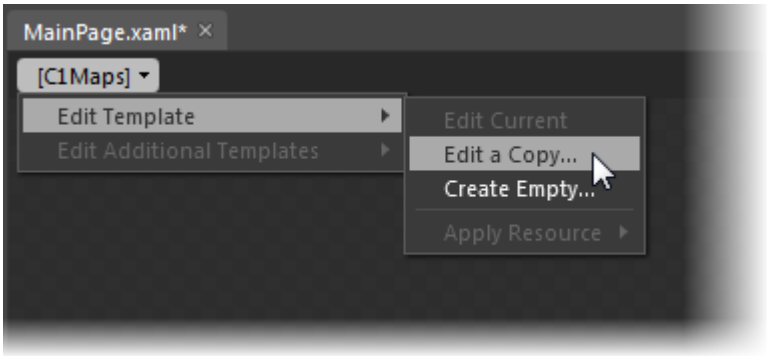
注意，只有当您第一次应用一个主题时该方法是有效的，如果您想切换其他的ComponentOne主题，首先需要将之前的主题从**Application.Current.Resources.MergedDictionaries**中移除。
通过在标签中添加主题声明，您可以为C1Maps控件中添加多个主题，更多C1Maps中添加主题相关信息，请参阅**使用C1Maps主题**。

模板

WPF及Silverlight版控件的一个优势在于控件具有完全自定义的"无外观"用户界面，例如您在为WPF或Silverlight应用设计用户界面（UI）时，可以通过**WPF及Silverlight版Maps来**实现适用于数据管理的UI。可扩展标记语言（XAML;发音为"Zammel"）是一种基于XML的声明式语言，它提供了一种简易的方式可以不用编写代码就可以完成您的UI设计。

可访问模板

您可以访问Microsoft Expression Blend获取模板，选择C1DockControl并在对象菜单中选择**编辑模板**，通过选择**编辑拷贝**来创建当前模板可编辑的一个拷贝，或者选择创建一个新的**空白模板**。



注意：如果您通过菜单方式创建了一个新的模板，模板将会自动的连接模板的属性，但如果您用XAML手动创建了一个模板，您就必须为您创建的模板提供适宜的模板属性。

注意: 您可以使用Template属性定制您的模板。

WPF及Silverlight版Maps任务帮助

如果您熟悉Visual Studio编程且了解在一般情况下如何使用C1Maps，请直接参考任务帮助，如果您是初次使用**WPF及Silverlight版Maps**产品，请先参阅**Silverlight及WPF版C1Maps快速入门**。

在使用WPF及Silverlight版Maps产品时，每个任务帮助主题都提供一个特定任务的解决方案。

在您参阅任务帮助主题前，请确定您已经创建了一个新的WPF或Silverlight项目。

添加一个标签

在本主题中，通过使用**C1VectorLayer及C1VectorPlacemark**，您将添加一个地理标签—宾夕法尼亚州伊利市（美国）地理坐标。了解更多矢量层相关信息，请参阅**矢量层**。

XAML

完成以下步骤：

1. 将下述XAML标签添加至`<c1:C1Maps>`与`</c1:C1Maps>`之间:

XMAL

```
<c1:C1VectorLayer>
<c1:C1VectorPlacemark LabelPosition="Left" GeoPoint="-80.107008,42.16389"
StrokeThickness="2" Foreground="#FFEB1212" PinPoint="-80.010866,42.156831"
Label="Erie, PA"/>
</c1:C1VectorLayer>
```

2. 运行项目。

代码

1. 在XAML视图的<c1:C1Maps>标签中添加x:Name="C1Maps1"， 这样项目则拥有一个在代码中被调用的唯一标识符。
2. 在代码框中输入或导入下述命名空间：

Visual Basic

```
Imports C1.Silverlight.C1Maps
```

C#

```
using C1.Silverlight.C1Maps;
```

3. 在 **InitializeComponent()**方法下添加下述代码：

Visual Basic

```
' 创建一个层并添加到地图中
Dim vl As C1VectorLayer = New C1VectorLayer()
C1Maps1.Layers.Add(vl)

' 创建一个矢量标记并添加到层中
Dim vp1 As C1VectorPlacemark = New C1VectorPlacemark()
vl.Children.Add(vp1)
```

```
' 为矢量标记设置一组地理坐标
vp1.GeoPoint = New Point(-80.107008, 42.16389)

' 设置标记的标签和属性
vp1.Label = "Erie, PA"
vp1.FontSize = 12
vp1.Foreground = New SolidColorBrush(Colors.Red)
vp1.LabelPosition = LabelPosition.Center
```

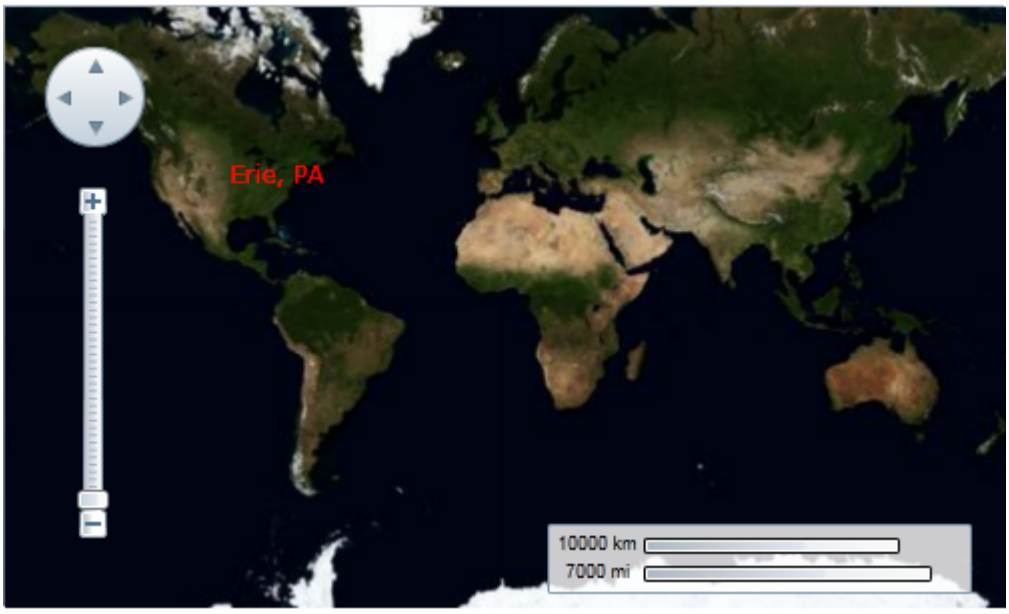
C#

```
//创建一个层并添加到地图中
C1VectorLayer vl = new C1VectorLayer();
C1Maps1.Layers.Add(vl);

//创建一个矢量标记并添加到层中
C1VectorPlacemark vp1 = new C1VectorPlacemark();
vl.Children.Add(vp1);

//为矢量标记设置一组地理坐标
vp1.GeoPoint = new Point(-80.107008, 42.16389);

//设置标记的标签和属性
vp1.Label = "Erie, PA";
vp1.FontSize = 12;
vp1.Foreground = new SolidColorBrush(Colors.Red);
vp1.LabelPosition = LabelPosition.Center;
```

4. 运行项目。

本主题示例如下图:

下方例图显示了使用**C1Maps**控件标记出宾夕法尼亚州伊利市（美国）的地理坐标。



添加一条折线

通过往**C1VectorLayer**中添加一个**C1VectorPolyline**（更多相关信息，请参阅**矢量层**），您可以以一条折线连接几个地理坐标，在本主题中，您使用XAML及代码创建一个包含三个点的折线。

XAML

完成如下步骤：

1. 将下述XAML标签置于<c1:C1Maps>与</c1:C1Maps>之间：

C#

```
<c1:C1VectorLayer Margin="2,0,-2,0">
     <c1:C1VectorPolyline Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
     </c1:C1VectorPolyline>
</c1:C1VectorLayer>
```

2. 按**F5**键运行项目。

代码

完成如下步骤：

1. 在XAML视图的<c1:C1Maps>标签中添加x:Name="C1Maps1"， 这样项目则拥有一个在代码中被调用的唯一标识符。
2. 在代码框中输入和导入下述命名空间：

Visual Basic

```
Imports C1.Silverlight.C1Maps
```

C#

```
using C1.Silverlight.C1Maps;
```

Visual Basic

```
' 创建一个层并添加到地图中
Dim C1VectorLayer1 As New C1VectorLayer()
C1Maps1.Layers.Add(C1VectorLayer1)
' 初始化轨迹
Dim pts As Point() = New Point() {New Point(-80.15, 42.12), New Point(-123.08,
39.09), New Point(-3.9, 30.85)}

' 创建一个集合并将坐标点添加到集合中
Dim pcoll As New PointCollection()

For Each pt As Point In pts
   pcoll.Add(pt)
Next

' 创建一条折线，并添加到矢量层中
Dim C1VectorPolyline1 As New C1VectorPolyline()
C1VectorLayer1.Children.Add(C1VectorPolyline1)
```

```
' 设置坐标点
C1VectorPolyline1.Points = pcoll

' 设置外观
C1VectorPolyline1.Stroke = New SolidColorBrush(Colors.Red)
C1VectorPolyline1.StrokeThickness = 3
```

3. 在**InitializeComponent()**方法下添加下述代码：

C#

```
//创建一个层并添加到地图中
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
c1Maps1.Layers.Add(C1VectorLayer1);

//初始化轨迹
Point[] pts = new Point[] { new Point(-80.15,42.12), new Point(-123.08,39.09),
new Point(-3.90,30.85)};

//创建一个集合并将坐标点添加到集合中
PointCollection pcoll = new PointCollection();
foreach( Point pt in pts)
pcoll.Add(pt);

//创建一条折线，并添加到矢量层中
C1VectorPolyline C1VectorPolyline1 = new C1VectorPolyline();
v1.Children.Add(C1VectorPolyline1);

//设置坐标点
   C1VectorPolyline1.Points = pcoll;

//设置外观
  C1VectorPolyline1.Stroke = new SolidColorBrush(Colors.Red);
  C1VectorPolyline1.StrokeThickness = 3;
```

4. 按**F5**键运行项目。

本主题示例如下图：

下图描绘了使用**C1Maps**控件将三个地理坐标点用一条折线连接起来。

添加一个多边形

通过往**C1VectorLayer**中添加一个**C1VectorPolygon**（更多相关信息，请参阅**矢量层**），您可以以一个多边形连接几个地理坐标，在本主题中，您使用XAML及代码创建一个包含三个点的多边形。

XAML

完成如下步骤：

1. 将下述XAML标签置于<c1:C1Maps>与</c1:C1Maps>之间：

XAML

```
<c1:C1VectorLayer Margin="2,0,-2,0">
     <c1:C1VectorPolygon Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
     </c1:C1VectorPolygon>
</c1:C1VectorLayer>
```

2. 按**F5**键运行项目。

代码

完成如下步骤：

1. 在XAML视图的<c1:C1Maps>标签中添加x:Name="C1Maps1"， 这样项目则拥有一个在代码中被调用的唯一标识符。
2. 在代码框中输入或导入下述命名空间：

Visual Basic

```
Imports C1.Silverlight.C1Maps
```

C#

```
using C1.Silverlight.C1Maps;
```

3. 在**InitializeComponent()**方法下添加下述代码：

Visual Basic

```
' 创建一个层并添加到地图中
Dim C1VectorLayer1 As New C1VectorLayer()
C1Maps1.Layers.Add(C1VectorLayer1)

' 初始化轨迹
Dim pts As Point() = New Point() {New Point(-80.15, 42.12), New Point(-123.08,
39.09), New Point(-3.9, 30.85)}

' 创建一个集合并将坐标点添加到集合中
Dim pcoll As New PointCollection()

For Each pt As Point In pts
   pcoll.Add(pt)
Next
```

```
' 创建一条折线，并添加到矢量层中

Dim C1VectorPolygon1 As New C1VectorPolygon()
C1VectorLayer1.Children.Add(C1VectorPolygon1)

' 设置坐标点


C1VectorPolygon1.Points = pcoll

' 设置外观


C1VectorPolygon1.Stroke = New SolidColorBrush(Colors.Red)
C1VectorPolygon1.StrokeThickness = 3
```

C#

```
//创建一个层并添加到地图中
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
C1Maps1.Layers.Add(C1VectorLayer1);

//初始化轨迹
Point[] pts = new Point[] { new Point(-80.15,42.12), new Point(-123.08,39.09),
new Point(-3.90,30.85)};

//创建一个集合并将坐标点添加到集合中
PointCollection pcoll = new PointCollection();
foreach( Point pt in pts)
pcoll.Add(pt);

//创建一条折线，并添加到矢量层中
C1VectorPolygon C1VectorPolygon1 = new C1VectorPolygon();
v1.Children.Add(C1VectorPolygon1);

//设置坐标点
   C1VectorPolygon1.Points = pcoll;

//设置外观
  C1VectorPolygon1.Stroke = new SolidColorBrush(Colors.Red);
  C1VectorPolygon1.StrokeThickness = 3;
```

4. 按**F5**键运行项目。

本主题示例如下图：

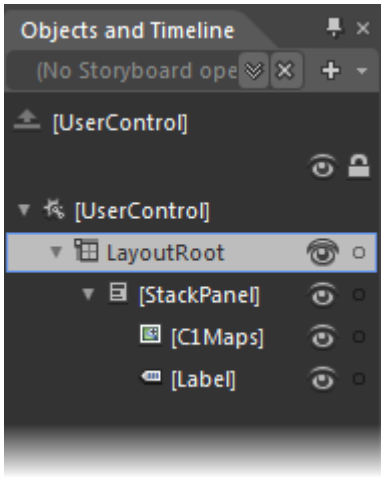
下图描绘了使用**C1Maps**控件将三个地理坐标点用一个多边形连接起来。

鼠标悬停地理坐标显示

在本主题中，您通过在项目中添加相应的代码，即可返回当前鼠标位置的地理坐标，这些地理坐标最终可以被写入**TextBox**控件中的**Text**属性。

为得到当前鼠标位置的地理坐标，需完成下列步骤：

1. 添加一个**StackPanel**，**TextBox** 及**C1Maps**控件到您的项目中。
2. 在**Objects and Timeline**面板中，重新排列控件，其显示如下图所示：



3. 按下述设置**StackPanel**属性：
   - 定位**Width**属性并点击图形符□，设置**Width**属性为**Auto**。
   - 定位**Height**属性并点击图形符□，设置**Height**属性为**Auto.**
4. 设置**TextBox**控件的**Name**属性为"ShowCoordinates"。
5. C1Maps control的属性设置如下：
   - **Width**属性设置为"350"。
   - **Height**属性设置为"250"。
6. 在**Properties**面板中选择C1Maps控件，并点击**Events**按钮。
7. 在**MouseMove**文本框中，输入"MouseMoveCoordinates"并按ENTER键将**MouseMoveCoordinates**事件处理加入到您的项目中。
8. 将代码注释用下述代码替换：

Visual Basic

```
Dim map As C1Maps = TryCast(sender, C1Maps)
Dim p As Point = map.ScreenToGeographic(e.GetPosition(map))
```

```
ShowCoordinates.Text = String.Format("{0:f6},{1:f6}", p.X, p.Y)
```

C#

```
C1Maps map = sender as C1Maps;
Point p = map.ScreenToGeographic(e.GetPosition(map));
ShowCoordinates.Text = string.Format("{0:f6},{1:f6}", p.X, p.Y);
```

9. 导入下述命名空间：

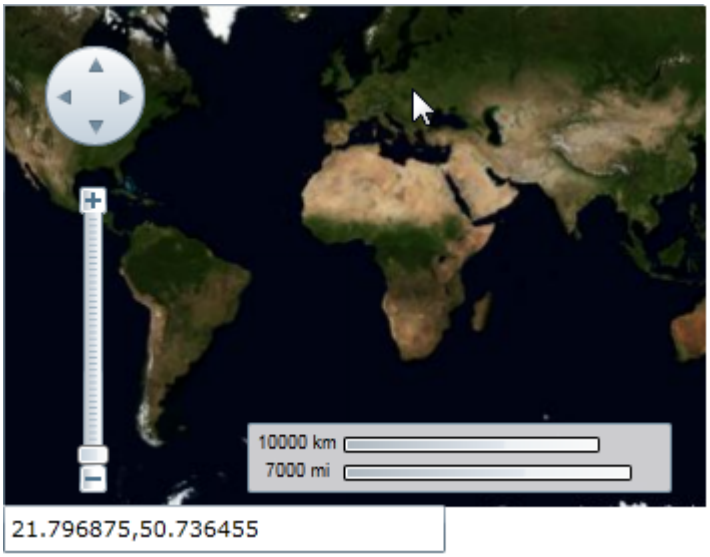
Visual Basic

```
Imports C1.WPF.C1Maps
```

C#

```
using C1.WPF.C1Maps;
```

10. 按**F5**键运行项目，一旦项目被载入，您可以看到当鼠标的光标在地图上悬停时，在文本框中会显示相应的地理坐标。



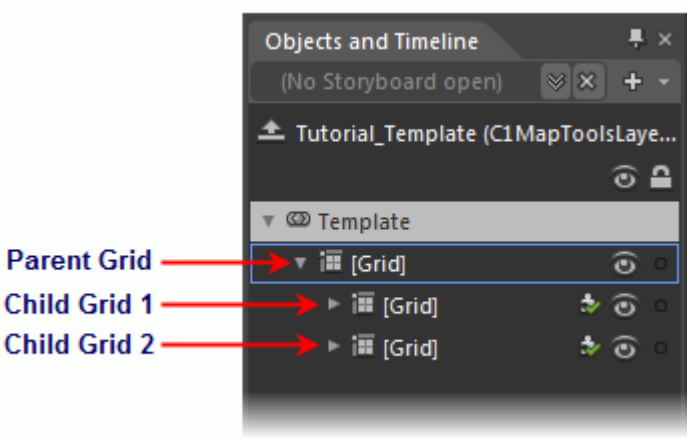
重新排列Map工具

您可以通过**C1MapToolsLayer**和模板来更改地图工具，了解更多详细信息，请参阅**自定义工具**。

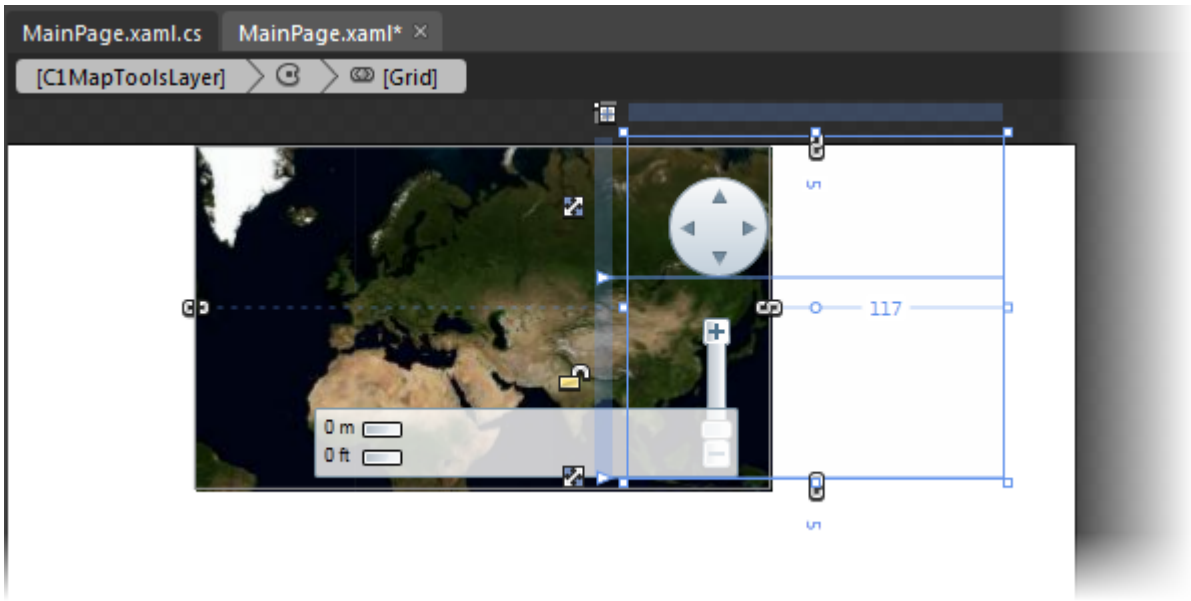
请完成如下步骤：

1. 选择C1Maps来显示Properties面板中的属性列表。
2. 清空**Show Tools** 中的复选框，此时会隐藏所有默认工具。
3. 点击**Layers (Collection)**省略号按钮， 此时**IMapLayer**集合编辑器中**Layers** 对话框将被打开。
4. 点击**Add another item**打开**Select Object** 对话框。
5. 选择**C1MapToolsLayer**并点击**OK**来关闭**Select Object** 对话框。
6. 在**Objects and Timeline**面板中，右键点击**[C1MapToolsLayer]**并选择**Edit Template | Edit a Copy**。将模板命名为"Tutorial Template"并点击**OK**。

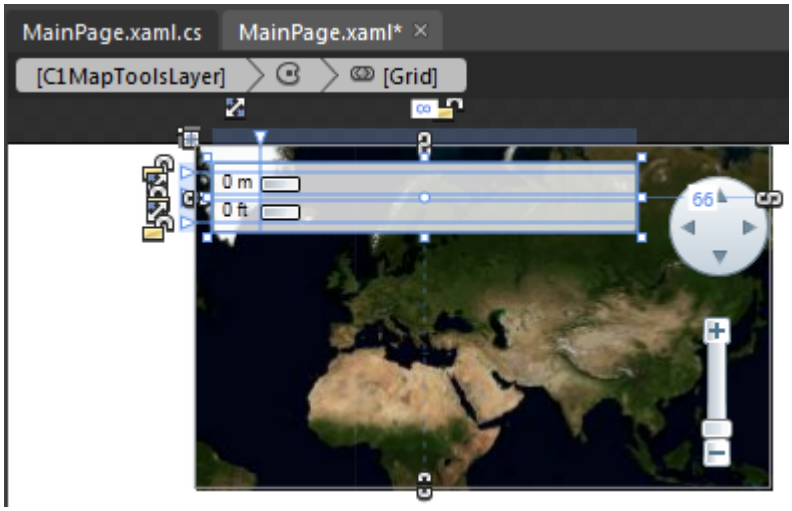
   此时您创建了一个新模板，观察下图中**Objects and Timeline**标签有一个父网格和两个子网格。



7. 点击**Child Grid 1** 可以在设计视图中显示它所表示的网格。
8. 在设计视图中，使用光标将已被选中的网格移动到地图的右侧，此时您的项目会与下图相似：



9. 在**Objects and Timeline**面板中，点击**Child Grid 2** 可以在设计视图中显示它所表示的网格。
10. 在设计视图中，使用光标将已被选中的网格移动到地图的左上方，此时您的项目会与下图相似：



11. 按**F5**键运行项目，此时C1Maps控件显示如下：

更改地图资源

C1Maps可以从不同的资源中显示地理信息，默认情况下，C1Maps采用Microsoft LiveMaps航拍图片作为资源，但您可以通过**Source**属性对资源类型做出更改，它是一个**MultiScaleTileSource**类型的对象。在默认情况下，它被设置为显示**Bing Map**（在使用此项服务之前请参阅**法律要求**）航拍图片，但您可以更改显示路拍资源或混合资源。

更改为路拍资源

完成下述步骤：

1. 在XAML视图的<c1:C1Maps>标签中添加x:Name="C1Maps1"， 这样项目则拥有一个在代码中被调用的唯一标识符。
2. 在代码框中输入或导入下述命名空间：

Visual Basic

```
Imports C1.Silverlight.C1Maps
```

C#

```
using C1.Silverlight.C1Maps;
```

3. 在**InitializeComponent()**方法下添加下述代码:

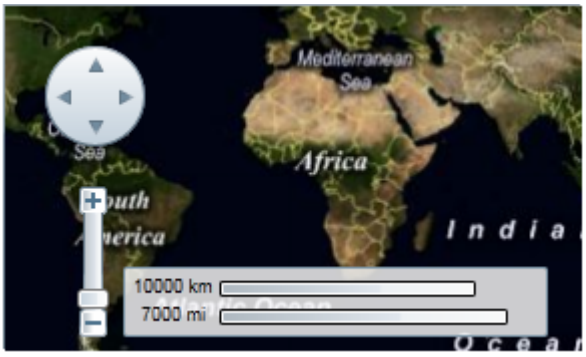
Visual Basic

```
C1Maps1.Source = new VirtualEarthRoadSource()
```

C#

```
C1Maps1.Source = new VirtualEarthRoadSource();
```

4. 按**F5**键运行程序，可以在下图中显示路拍地图资源。



更改为混合资源

完成以下步骤：

1. 在XAML视图的<c1:C1Maps>标签中添加x:Name="C1Maps1"， 这样项目则拥有一个在代码中被调用的唯一标识符。
2. 在代码框中输入或导入下述命名空间：

Visual Basic

```
Imports C1.Silverlight.C1Maps
```

C#

```
using C1.Silverlight.C1Maps;
```

3. 在**InitializeComponent()**方法下添加下述代码:

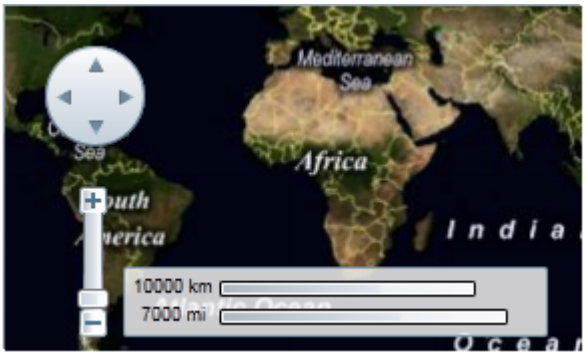
Visual Basic

```
C1Maps1.Source = new VirtualEarthHybridSource()
```

C#

```
C1Maps1.Source = new VirtualEarthHybridSource();
```

4. 按**F5**键运行程序，可以在下图中显示混合地图资源。



了解更多地图资源的详细信息，请参阅**C1Maps概念和主要属性**。

使用C1Maps主题

C1Maps控件默认配置一个浅蓝色主题，但您可以为控件应用六种不用的主题（请参考**C1Maps主题**），在本主题中，您将通过**C1ThemeRainerOrange**对**C1Maps**控件的主题进行更改。

Blend中设置

完成以下步骤：

1. 点击**Assets**标签。
2. 在搜索栏中输入"C1ThemeRainierOrange"，此时会出现**C1ThemeRainierOrange**的图标。
3. 双击**C1ThemeRainierOrange**图标添加到您的项目中。
4. 在搜索栏中输入"C1Maps" 查找到**C1Maps**控件。
5. 双击**C1Maps**图标添加**C1Maps**控件到您的项目中。

6. 在**Objects and Timeline**面板中，选择**[C1Maps]**并使用拖放操作将C1Maps置于**[C1ThemeRainierOrange]**下。
7. 运行项目。

Visual Studio中设置

完成以下步骤：

1. 在Visual Studio中打开 **.xaml** 页面。
2. 将您的光标置于*<Grid></Grid>* 标签之间。
3. 在In**Tools**面板中，双击**C1ThemeRainierOrange** 图标声明主题，其标签会显示如下：

   <my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>

4. 将光标置于*<my:C1ThemeRainierOrange>*与*</my:C1ThemeRainierOrange>* 标签之间。
5. 在**Tools**面板中，双击C1Maps图标将控件添加到项目中，此时会出现*<my:C1ThemeRainierOrange>* 的子标签，该标签与下述描述相似：

XAML

```
<my:C1ThemeRainierOrange>
    <c1:C1Maps Height="172" Width="288" Margin="200,0,34,0"/>
</my:C1ThemeRainierOrange>
```

6. 运行项目。

本主题示例如下图：

下图显示了配置**C1ThemeRainierOrange**主题的**C1Maps**控件。