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
允许的方案允许的方案	HTTP，HTTPS	HTTP，HTTPS，FILE	HTTP，HTTPS，FILE	HTTP
跨方案访问跨方案访问	否	否	否	HTTPS以外
跨跨Web领域访问领域访问	否	HTTPS以外	否	是
跨区域访问跨区域访问(Windows)	否	否	否	否
跨区域访问跨区域访问(Macintosh)	否	是	否	是
允许重定向允许重定向	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访问策略访问策略：http://msdn.microsoft.com/enus/library/bb820909.aspx.

注意注意：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();
```

虚拟地球混合资源虚拟地球混合资源

```
Visual Basic
map3.Source = new VirtualEarthHybridSource()
C#
map3.Source = new VirtualEarthHybridSource();
```

可视化地图可视化地图
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
{
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="4a3704ea-e789-4fd3-98bb-b27e2a903948"><ac:plain-text-body><![CDATA[ ItemsSource = new[]
]]></ac:plain-text-body></ac:structured-macro>
{
```

```
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。
数据绑定示例数据绑定示例
假设您有一个城市城市对象集：

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#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="fba13bb8-c946-4dbd-b77e-8f413f1200da"><ac:plain-text-body><![CDATA[
```

```
City[] cities = new City[]
]]></ac:plain-text-body></ac:structured-ma
{ new City(){ LongLat= new Point(30.32,59.
, Name="Saint Petersburg"}, new City(){
LongLat= new Point(24.94,60.17), Name="Hel
ki"}, new City(){ LongLat= new Point(18.07
.33), Name="Stockholm"}, new City(){ LongL
new Point(10.75,59.91), Name="Oslo"}, new
y(){ LongLat= new Point(12.58,55.67), Name
openhagen"}
}; maps.DataContext = cities;
```

自定义工具

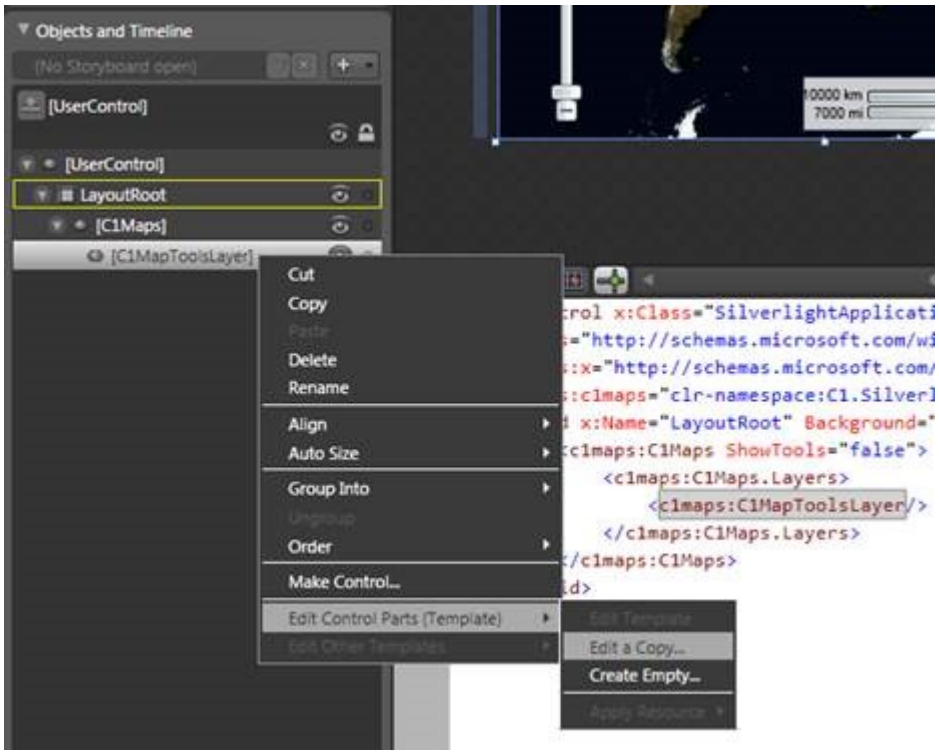
平移及缩放工具默认在地图中显示，并通过C1MapToolsLayer实现，然而它被集成在C1Maps的模板中，所以我们不需要将它加入到图层集合中。为了实现工具的自定义，您需要先把C1Maps.ShowTools置为False，此时工具被设置为隐藏，此后再为C1MapToolsLayer添加实例，以下是相关的XAML：

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）编辑控件（模板）|编辑拷贝编辑拷贝：



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。