Maps for UWP

2018.03.07 更新

グレープシティ株式会社

Maps for UWP

目次

Maps for UWP

Maps for UWP は、スムーズなズームとパン、および画面と地理座標の間のマッピングにより、画像表示のレベルを大幅に向上させます。**C1Maps** を使用すると、Bing Maps から取得した豊富な地理情報を表示できます。

Microsoft Deep Zoom テクノロジー上に構築された **C1Maps** を使用して、エンドユーザーは、高解像度の画像とスムーズなジャンプによる究極のクローズアップ操作を楽しむことができます。また、マップに独自のカスタム要素を重ねるためのレイヤもサポートされます。

主な特長

Maps for UWP を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次の主要な機能を利用して、**Maps for UWP** を最大限に活用してください。

- **ジオメトリの描画**
  C1Maps のベクターレイヤを使用すると、地理座標を持つジオメトリ、図形、多角形、パスをマップ上に描画することができます。ベクターレイヤは、次を項目の描画に役立ちます。
  - 行政界（国境、県境など）
  - 地理情報（路線図、航路図の表示など）
  - コロプレスマップ（国別の人口などの統計データを示すマップ）

  Microsoft Virtual Earth の通常のソースの代わりにベクターレイヤを使用して、世界地図を表示できます。
- **KML サポート**
  ベクターレイヤは、基本的な KML のインポート/エクスポート機能をサポートします。KML は、マップ上の描画を交換するための標準ファイル形式です。詳細については、「KML インポート/エクスポート」を参照してください。
- **豊富な地理情報**
  Bing Map やカスタムソースなどのさまざまなソースから、豊富な地理情報を表示できます。たとえば、Yahoo! Maps 用の独自のソースを構築できます。
- **マップに多数の要素を表示可能**
  Maps for UWP では、ローカルデータやサーバーデータを仮想化することができます。仮想レイヤマップを使用することにより、現在表示可能な要素のみを表示したり要求することができます。
- **パン、マップ座標**
  Maps for UWP では、マウスまたはタッチを使ってパンを行うことができます。また、画面と地理座標の間のマッピングもサポートされています。
- **レイヤのサポート**
  レイヤを使用して、カスタム要素をマップに追加できます。要素は地理的な位置にリンクされます。詳細については、「ベクターレイヤ」、「仮想化」、および「項目のレイヤー」を参照してください。
- **DirectXのサポート**
  Direct X のサポートで**C1Maps**コントロールにスムーズなパンニングやズームを提供します。

クイックスタート

このクイックスタートガイドは、**Maps for UWP** を初めて使用するユーザーのために用意されています。最初に、Visual Studio で **C1Maps** コントロールを含む Windows ストアアプリケーションを作成します。コントロールを追加したら、その外観をカスタマイズし、**C1VectorLayer** および **C1VectorPlacemark** をコントロールに追加します。次に、データソースを作成し、C1VectorPlacemark のプロパティをデータソースに連結します。このクイックスタートの手順を最後まで実行すると、一連のラベル付きプレースマークを含む、必要な機能をすべて備えたマップコントロールが完成します。

手順1:アプリケーションの作成

この手順では、最初に Visual Studio で **C1Maps** コントロールを使用する Windows ストアアプリケーションを作成します。また、コントロールのプロパティも設定します。

次の手順を実行します。

1. [**ファイル**]→[**新規作成**]→[**プロジェクト**]を選択し、[**新しいプロジェクト**]ダイアログボックスを開きます。
    1. 右側のペインで C# の下にある[**Windows ストア**]を選択します。
    2. 左側のペインで[**新しいアプリケーション (XAML)**]を選択します。
    3. アプリケーションの名前を入力し、[**OK**]をクリックします。新しい空の Windows ストアアプリケーションが開きます。
2. ソリューションエクスプローラーで、[**参照**]ファイルを右クリックし、リストから[**参照の追加**]を選択します。次のアセンブリ参照を参照して選択します。
    - C1.Xaml.dll
    - C1.Xaml.Maps.dll
    - C1.Xaml.Zip.dll
3. **MainPage.xaml** ファイルをダブルクリックして開きます。
4. ページ先頭の `<Page>` タグに次の名前空間宣言を追加します。
    - xmlns:C1="using:C1.Xaml.Maps"
    - xmlns:Xaml="using:C1.Xaml"

    タグは次のようになります。

    XAML マークアップ

    ```
    lt;Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:C1="using:C1.Xaml.Maps"
    xmlns:local="using:MapsTest5"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Xaml="using:C1.Xaml"
    x:Class="MapsTest5.MainPage"
    mc:Ignorable="d">
    ```

5. 次のマークアップを `<Grid> </Grid>` タグの間に挿入して、C1Maps コントロールを追加します。

    XAML マークアップ

    ```
    <C1:C1Maps x:Name="maps" Foreground="LightGreen"></C1:C1Maps>
    ```

この手順では、新しい Windows ストアプロジェクトを作成し、C1Maps コントロールを追加しました。次の手順では、データソースにマップをバインドします。

手順2:データソースへ連結する

この手順では、**Name** と **LatLong** の2つのプロパティを持つクラスを作成し、配列コレクションを使ってそれらのプロパティに値を入力します。また、**C1VectorPlacemark** を含む **C1VectorLayer** をコントロールに追加します。次に、**Name** プロパティを C1VectorPlacemark の **Label** プロパティに連結し、**LatLong** プロパティを C1VectorPlacemark の **GeoPoint** プロパティに連結します。

次の手順に従います。

1. **MainPage.xaml** コードページ（**MainPage.xaml.cs または MainPage.xaml.vb**）を開きます。このページの拡張子は、プロジェクトに選択した言語によって異なります。
2. 次のクラスをプロジェクト内の名前空間宣言の下に追加します。
このクラスは、**Name** という名前の文字列プロパティおよび **LongLat** という名前の **Point** プロパティを含むクラスを作成します。

Visual Basic コードの書き方

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# コードの書き方

C#

```
public class City
   {
     public Point LongLat { get; set; }
     public string Name { get; set; }
     public City(Point location, string cityName)
        {
             this.LongLat = location;
```

```
            this.Name = cityName;
        }
    }
```

3. 次のコードを **InitializeComponent()** メソッドの下に追加して、**Name** プロパティおよび **LongLat** プロパティに入力される配列コレクションを作成します。

Visual Basic コードの書き方

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")
}
maps.DataContext = cities
```

C# コードの書き方

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"},
};
maps.DataContext = cities;
```

4. XAML ビューに切り替えて、`<c1:C1Maps>` タグと `</c1:C1Maps>` タグの間に次のような XAML マークアップを配置します。

XAML マークアップ

```
 <C1:C1Maps.Resources>
 <!-- 項目テンプレート -->
     <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"
/>
```

この XAML は、データテンプレート、C1VectorPlacemark、および C1VectorLayer を作成します。**C1VectorLayer** の ItemsSource プロパティがデータソース全体に連結されます。また、**C1VectorPlacemark** のGeoPoint プロパティは **LongLat** プロパティの値に連結され、Label プロパティは **Name** プロパティの値に設定されます。プロジェクトを実行すると、Label プロパティおよび Name プロパティにデータソースから値が入力されて、一連のラベル付きプレースマークがマップ上に作成されます。

この手順では、データソースを作成し、それを **C1VectorPlacemark** のプロパティに連結しました。次の手順では、プログラムを実行して、このクイックスタートガイドで作成したプロジェクトの結果を表示します。

手順3:アプリケーションの実行

前のステップでは、C1Maps コントロールを含む Windows ストアプロジェクトを作成し、データソースを作成し、C1VectorLayer および C1VectorPlacemark を **C1Maps** コントロールに追加しました。その後、データソースを **C1VectorPlacemark** のプロパティに連結しました。

次の手順に従います。

1. [F5]キーを押してプロジェクトを実行し、次のように C1Maps コントロールが表示されることを確認します。

カラカスの近くに2つ点があり、その横には名前がないことを確認します。

2. **カラカス**の近くをダブルクリックします。これを2回繰り返して、**ジョージタウン**と**パラマリボ** という2つのラベルが表示されることを確認します。

おめでとうございます。これで、**Maps for UWP** クイックスタートは終了です。ヘルプファイルの「**Maps コントロールの基本**」を参照して、コントロールに関する理解をさらに深めることをお勧めします。

法的要件

C1Maps では、Bing Maps から取得した地理情報を使用できます。このサービスを使用する前に、サービスの使用条件を確認する必要があります。使用条件は次のサイトで確認できます。

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

C1Maps を使用する Bing Map の認証方法

C1Maps が提供するマップサーフェス（VirtualEarthRoadSurface、VirtualEarthAerialSurface、VirtualEarthHybridSurface）は、Bing Maps API を使用します。これらのオンラインタイルソースを商用利用する場合は有料です。Bing Maps の使用方法の詳細については、以下の Microsoft ドキュメントを参照してください。

http://msdn.microsoft.com/ja-jp/library/dd877180.aspx

C1Maps を使用する Bing Maps アプリケーションの認証には、コードに Bing Maps Key を使用する方法をお勧めします。Bing Maps Key には有効期限がなく、キーの取得にサービス要求は必要ありません。そのため、Bing Maps Key と C1Maps コントロールを使用するアプリケーションの認証は極めて簡単です。

Bing Maps Key の取得

C1Maps コントロールがサポートする組み込みのタイルソースを使用する場合は、まず、Bing Maps Key を取得する必要があります。Bing Maps Account Center にアクセスし、アカウントを作成してからキーを取得します。

http://www.bingmapsportal.com/

キーの取得の詳細については、以下の Microsoft ドキュメントを参照してください。

http://msdn.microsoft.com/ja-jp/library/ff428642.aspx

C1Maps の使用の認証

キーを取得したら、キーをコンストラクタに渡すことで、提供されている Bing Maps タイルソース（VirtualEarthRoadSurface、VirtualEarthAerialSurface、VirtualEarthHybridSurface）のいずれの使用も認証できます。次に例を示します。

C#

```
string credentialsProvider = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
C1MultiScaleTileSource tileSource = new VirtualEarthRoadSource(credentialsProvider);
myMap.Source = tileSource
```

XAML クイックリファレンス

このトピックでは、さまざまな C1Maps タスクの実行に使用される XAML の概要を提供します。

項目テンプレート

次の例は、マップ項目テンプレートを使用する方法を示します。

XAML マークアップ

```
 <C1:C1Maps x:Name="C1Maps1" FadeInTiles="False" Margin="0,0,235,8" TargetCenter="-
65,
        -25" Center="-58,-25" Zoom="2" Foreground="Aqua" Xaml:C1NagScreen.Nag="True">
           <C1:C1Maps.Resources>
             <!-- 項目テンプレート -->
                <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:C1Maps>
```

ベクターレイヤラベル

次の例は、ベクターレイヤを使用してラベルを作成する方法を示します。

XAML マークアップ

```
<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>
```

ベクターレイヤ– 折れ線

次の例は、ベクターレイヤを使用して、折れ線(開いた線)を作成する方法を示します。

```
<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>
```

ベクターレイヤ– 多角形

次の例は、ベクターレイヤを使用して、折れ線(形状を成す線)を作成する方法を示します。

```
<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>
```

マップの概念と主要なプロパティ

ここでは、基本的なマップの概念と主要なプロパティについて詳しく説明します。

マップソース

C1Maps は、さまざまなソースから取得した地理情報を表示できます。デフォルトでは、**C1Maps** はソースとして **Microsoft LiveMaps** の航空写真を使用しますが、**Source** プロパティを使ってソースを変更することもできます。このプロパティは **MultiScaleTileSource** 型のオブジェクトです。

次のソースがあります。

- Virtual Earth Aerial ソース

VisualBasic

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

C#

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

- Virtual Earth Road ソース


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

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

- Virtual Earth Hybrid ソース


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

C#

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

表示されるマップ

マップとして現在表示されている部分は、**Center** プロパティと **Zoom** プロパティ、およびコントロールのサイズによって決定されます。

- **Center** プロパティは **Point** 型ですが、実際は地理座標を表し、X プロパティが経度、Y プロパティが緯度です。ユーザーは、マップをドラッグして、**Center** プロパティの値を変更できます。

マークアップ

```
<Maps:C1Maps Name="maps" Center="-58.40, -34.36"/>
```

C#

```
maps.Center = new Point(-58.40, -34.36);
```

- **Zoom** プロパティは、マップの現在の解像度を指定します。ズーム値0でマップは完全にズームアウトし、値が1増えるたびにマップの解像度が倍になります。

マークアップ

```
<Maps:C1Maps Name="maps" MinZoom= "5" MaxZoom= "15"/>
```

C#

```
maps.MinZoom = 5;
maps.MaxZoom = 15;
```

座標系

C1Maps は、次の3つの座標系を使用します。

- **地理座標**は、緯度と経度を使って地球上の各地点を示します。この座標系はデカルト座標ではありません。つまり、パンしてもマップのスケールは変わりません。
- **論理座標**は、各軸の0～1の値でマップの全範囲を表します。デカルト座標なので、操作が簡単です。
- **画面座標**は、左上隅を基準にしたコントロールのピクセル座標です。これは、コントロール内の項目の配置やマウスイベントの処理に使用されます。

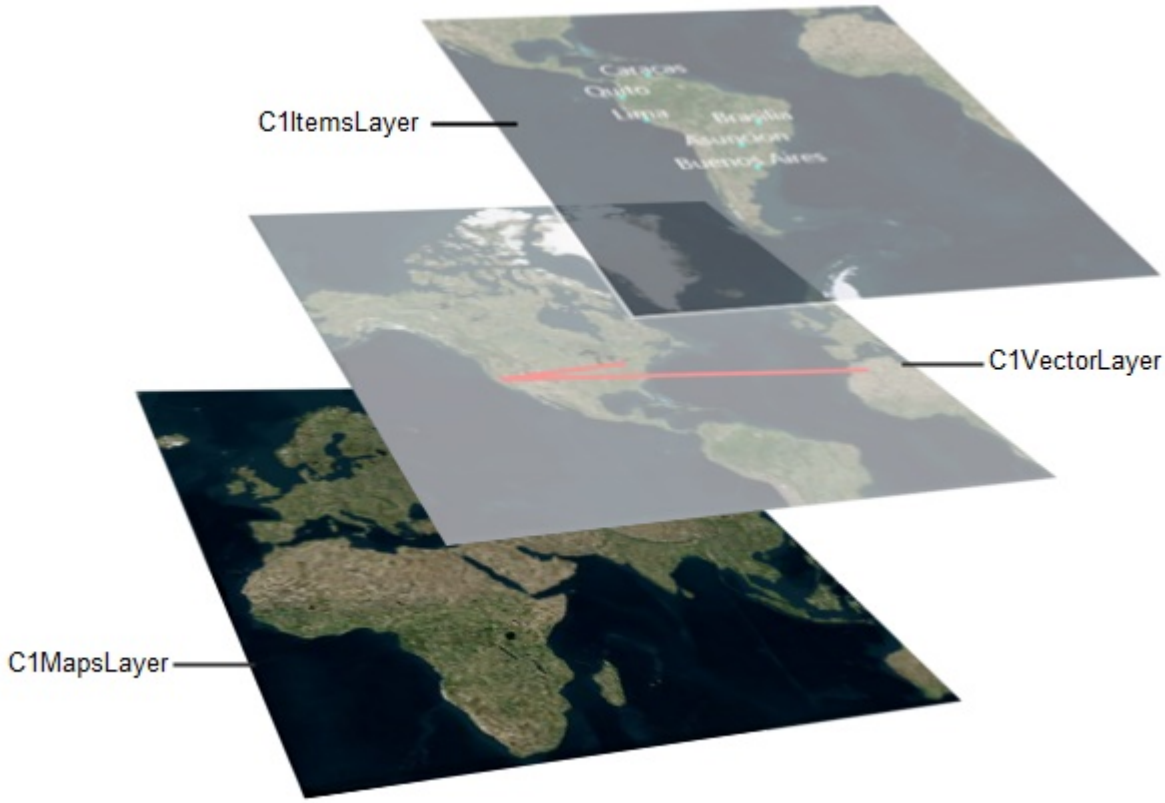
C1Maps には、これらの座標系間の変換に使用される **ScreenToGeographic**、**ScreenToLogic**、**GeographicToScreen**、**LogicToScreen** の4つのメソッドが用意されています。地理座標と論理座標の間の変換は、**C1Maps.Projection** プロパティを使って設定される投影法に基づいて行われます。投影法は、異なるマップをサポートするように変更できます。デフォルトは、**LiveMaps** などのほとんどのプロバイダが使用しているメルカトル図法です。

情報レイヤ

ソースから提供される地理情報のほかに、情報レイヤをマップに追加できます。**C1Maps** には、デフォルトで次の5つのレイヤ

が用意されています。

次の画像では、最も使用する3層を示します。



- **C1MapItemsLayer** は、任意の項目をマップ上に地理的に配置して表示するために使用されるレイヤです。このレイヤは **ItemsControl** なので、**UIElement** オブジェクトまたは汎用データオブジェクトの追加を直接サポートします。また、それらのオブジェクトをビジュアル項目に変換するために、**DataTemplate** が使用されます。
- **C1MapVirtualLayer** は、仮想化された項目を表示します。つまり、それらの項目は、それが属するマップ領域が表示されている場合にのみロードされます。また、非同期リクエストがサポートされるため、新しい項目が表示されたときにのみ、サーバーからそれらの項目がダウンロードされます。
- **C1VectorLayer** は、その頂点が地理的に配置されている直線や多角形などのベクターデータを表示します。KML ファイルの間でデータを保存したりロードすることができます。
- **C1MapTilesLayer** は、マップタイルが表示される背景レイヤです。このレイヤは C1Maps によって自動的に管理されるため、通常、これを使用する必要はありません。

C1Mapsの使い方

項目のレイヤー

C1MapItemsLayer は、マップ上に項目を表示する最も簡単な方法です。これは **ItemsControl** を継承するため、**UIElement** オブジェクトまたは汎用データオブジェクトの追加を直接サポートします。また、それらのオブジェクトをビジュアル項目に変換するために、**DataTemplate** が使用されます。**C1MapItemsLayer** に追加される要素は、**C1MapCanvas.LatLong** 添付プロパティを使って配置されます。サンプルを紹介します。

XAML マークアップ

```
<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>
```

これは、XAML で **C1Maps** コントロールを作成し、その **Layers** コレクションに **C1MapItemsLayer** を追加します。**Layers** コレクションにはレイヤをいくつでも追加でき、それらのレイヤは重ねて表示されます。

この項目レイヤには、1つの項目として、緯度/経度（40.4587, -79.9247）の位置に楕円を追加しています。これらの数値は、XAML では逆になっていることに注意してください。これは、**LatLong** 値が、経度に対応する X 値と緯度に対応する Y 値から成る **Point** 構造体で表されるためです（これは、マップと X/Y 軸の通常の配置に一致しています）。

この例では、**C1MapCanvas.Pinpoint** 添付プロパティが使用されていることもわかります。このプロパティは、要素上のどのポイントを **LatLong** プロパティで設定された地理座標に置くかを設定します。この例では、楕円の中央が LatLong 位置に来るように、**Pinpoint** が（10, 10）に設定されています。

2番目の例を紹介します。今回は、コードで **C1Maps** コントロールを作成し、そのコントロールにデータを設定します。次のクラスを使用します。

C#

```
public class Place
        {
            public string Name { get; set; }
            public Point LatLong { get; set; }
        }
```

サンプルコードは次のとおりです。

```
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);
```

ItemsSource に **Place** クラスのインスタンスを格納し、**ItemTemplate** を Page のリソースで定義された次の **DataTemplate** に設定します。

XAML マークアップ

```
<DataTemplate x:Name="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>
```

この **DataTemplate** は、**C1MapCanvas.LatLong** を項目で定義された **LatLong** にバインドし、**TextBlock** に Place の Name を表示します。

ItemTemplate と **ItemsSource** を使用すると、データベースから簡単にデータをロードできます。データオブジェクトのコレクションを返す Web サービスを設定し、コレクションを **ItemsSource** として設定し、適切な値をバインドする **DataTemplate** を作成するだけです。

オフラインマップ

Maps for UWP を使用すると、簡単にオフラインマップを利用できます。

このサンプルには、次のカスタム **OfflineMapsSource** クラスの実装が含まれています。

C#

```
    public class OfflineMapsSource : C1MultiScaleTileSource
{
private const string uriFormat = @"ms-appx:/Resources/OfflineMaps/
        {Z}/{X}/{Y}.png";

public OfflineMapsSource()
: base(0x8000000, 0x8000000, 0x100, 0x100, 0)
{ }

protected override void GetTileLayers(int tileLevel, int tilePositionX,
        int tilePositionY, IList<object> source)
{
if (tileLevel > 8)
{
var zoom = tileLevel - 8;
var uri = uriFormat;uri = uri.Replace("{X}", tilePositionX.ToString());
uri = uri.Replace("{Y}", tilePositionY.ToString());
uri = uri.Replace("{Z}", zoom.ToString());
```

```
source.Add(new Uri(uri));
}
}
}
```

この実装クラスは、ローカルの Resource フォルダからタイル画像をロードします。このクラスは C1MultiScaleTileSource を継承します。

オフラインマップでカスタムタイルを使用するには、C1Maps の Source プロパティを設定する必要があります。このサンプルでは、**OnMapsLoaded** イベントが作成され、このイベント内で Source プロパティが設定されます。

C#

```
void OnMapsLoaded(object sender, RoutedEventArgs e)
{
this.maps.Source = new OfflineMapsSource();
}
```

このように、オフライン C1Maps コントロールの作成は簡単です。

仮想化

C1MapVirtualLayer は、仮想化と非同期データロードをサポートしてマップ上に要素を表示します。一度に表示する要素が多くない場合は、これを使用して、無制限の数の要素を表示できます。そのオブジェクトモデルは **C1MapItemsLayer** とはまったく異なります。**C1MapVirtualLayer** では、マップ空間がいくつかの領域に分割されている必要があり、項目のソースは **IMapVirtualSource** インターフェイスを実装している必要があります。

マップ空間の分割は、**MapSlice** の **C1MapVirtualLayer.Slices** コレクションを使って定義されます。各マップスライスはに、その区分の最小ズームレベルが定義され、あるスライスの最大ズームレベルが次のスライスの最小ズームレベルになります。したがって、最後のスライスの最大ズームレベルは、マップの最大ズームレベルになります。さらに、各スライスは、緯度/経度のグリッドに分割されます。

例として次のレイヤを紹介します。

C#

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

ここには、ズーム5～10 とズーム10～最大ズームの2つのスライスがあります。ズーム値が元のスライスから別のスライスに移動すると、仮想レイヤはそのソースにデータを要求します。また、最初のスライスは緯度/経度によって2×2に分割されます。つまり、マップは4つの領域に分割され、レイヤは現在表示されている領域のデータのみを要求します。2番目のスライスが 16 個の領域に分割されているのは、ズーム値が大きくなるほど多数に分割した方がパフォーマンスが向上するためです。

IMapVirtualSource インターフェイスを理解するために、Factories サンプルの実装を紹介します。

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 メソッドは、マップ空間の1つの領域をパラメータとして受け取るほか、コールバックを使って返される項目のコレクションを受け取ります。この実装は最初に、要求された最小ズームがアプリケーションパラメータより小さいかどうかをチェックし、小さい場合は何もしません。そうでない場合は、Web サービスを呼び出してデータを取得します。

サーバー側には **GetStores** の実装があります。これは、データベース内のすべての要素を反復処理して、要求された範囲内にある項目を返します。

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;
}
```

さらに上手な実装としては、すべての項目を反復処理しなくても済むように、あらかじめストアを領域に分割しておきます。

ベクターレイヤ

ベクターレイヤを使用して、マップ上の地理座標でさまざまなオブジェクトを配置できます。

ベクターオブジェクト

ベクターレイヤで使用できる主要なベクター要素を次に示します。

C1VectorPolyline

このトピックでは、**C1VectorLayer** および **C1VectorPlacemark** を使用して、ラベルを地理的位置（米国 Pennsylvania 州 Erie の地理座標）に追加します。ベクターレイヤの詳細については、「ベクターレイヤ」を参照してください。

XAML の場合

次の手順に従います。

1. 次の XAML を <c1:C1Maps> タグと </c1:C1Maps> タグの間に追加します。

XAML マークアップ

```
<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. プロジェクトを実行します。

コードの場合

1. XAML ビューで、x:Name="C1Maps1" を <c1:C1Maps> タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. コードビューに切り替えて、次の名前空間をインポートします。

Visual Basic

```
Imports C1.Xaml.Maps
```

C#

```
using C1.Xaml.Maps;
```

3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic コードの書き方

Visual Basic

```
' レイヤを作成してマップに追加します
Dim vl As C1VectorLayer = New C1VectorLayer()
C1Maps1.Layers.Add(vl)

'ベクタープレースマークを作成してレイヤに追加します
Dim vp1 As C1VectorPlacemark = New C1VectorPlacemark()
vl.Children.Add(vp1)

' プレースマークを一連の地理座標に設定します
vp1.GeoPoint = New Point(-80.107008, 42.16389)

' プレースマークのラベルとプロパティを設定します
vp1.Label = "Erie, PA"
```

```
vp1.FontSize = 12
vp1.Foreground = New SolidColorBrush(Colors.Red)
vp1.LabelPosition = LabelPosition.Center
```

C# コードの書き方

C#

```
// レイヤを作成してマップに追加します
C1VectorLayer vl = new C1VectorLayer();
C1Maps1.Layers.Add(vl);

//ベクタープレースマークを作成してレイヤに追加します
C1VectorPlacemark vp1 = new C1VectorPlacemark();
vl.Children.Add(vp1);

// プレースマークを一連の地理座標に設定します
vp1.GeoPoint = new Point(-80.107008, 42.16389);

// プレースマークのラベルとプロパティを設定します
vp1.Label = "Erie, PA";
vp1.FontSize = 12;
vp1.Foreground = new SolidColorBrush(Colors.Red);
vp1.LabelPosition = LabelPosition.Center;
```

4. プロジェクトを実行します。

このトピックの作業結果

次の図は、米国 Pennsylvania 州 Erie のラベルが付いた地理座標を含む C1Maps コントロールを示しています。

C1VectorPolygon

地理座標を多角形で結ぶには、**C1VectorPolygon** を C1VectorLayer に追加します。詳細については、「ベクターレイヤ」を参照してください。このトピックでは、XAML またはコードを使用して、3つの点から成る多角形を作成します。

XAML の場合

次の手順に従います。

1. 次の XAML マークアップを <c1:C1Maps> タグと </c1:C1Maps> タグの間に配置します。

XAML マークアップ

```
<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. [F5]キーを押してプロジェクトを実行します。

コードの場合

次の手順に従います。

1. XAML ビューで、x:Name="C1Maps1" を <c1:C1Maps> タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. コードビューに切り替えて、次の名前空間をインポートします。

Visual Basic

```
Imports C1.Xaml.Maps
```

C#

```
using C1.Xaml.Maps;
```

3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic コードの書き方

Visual Basic

```
' レイヤを作成してマップに追加します
Dim C1VectorLayer1 As New C1VectorLayer()
C1Maps1.Layers.Add(C1VectorLayer1)

' 初期経路
Dim pts As Point() = New Point() {New Point(-80.15, 42.12), New Point(-123.08,
39.09), New Point(-3.9, 30.85)}

' コレクションを作成し、データを設定します
Dim pcoll As New PointCollection()
For Each pt As Point In pts
   pcoll.Add(pt)
Next

' 多角形を作成して、ベクターレイヤに子として追加します
Dim C1VectorPolygon1 As New C1VectorPolygon()
C1VectorLayer1.Children.Add(C1VectorPolygon1)
```

```
' ポイント
C1VectorPolygon1.Points = pcoll

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

C# コードの書き方

C#

```
// レイヤを作成してマップに追加します
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
C1Maps1.Layers.Add(C1VectorLayer1);

// 初期経路
Point[] pts = new Point[] { new Point(-80.15,42.12), new Point(-123.08,39.09),
new Point(-3.90,30.85)};

// コレクションを作成し、データを設定します
PointCollection pcoll = new PointCollection();
foreach( Point pt in pts)
pcoll.Add(pt);

// 多角形を作成して、ベクターレイヤに子として追加します
C1VectorPolygon C1VectorPolygon1 = new C1VectorPolygon();
C1VectorLayer1.Children.Add(C1VectorPolygon1);

// ポイント
        C1VectorPolygon1.Points = pcoll;

// 外観
  C1VectorPolygon1.Stroke = new SolidColorBrush(Colors.Red);
  C1VectorPolygon1.StrokeThickness = 3;
```

4. [F5]キーを押してプロジェクトを実行します。

このトピックの作業結果

次の図は、多角形で結ばれた3つの地理座標を含む C1Maps コントロールを示しています。

C1VectorPlacemark

このトピックでは、C1VectorLayer および **C1VectorPlacemark** を使用して、ラベルを地理的位置（米国 Pennsylvania 州 Erie の地理座標）に追加します。ベクターレイヤの詳細については、「ベクターレイヤ」を参照してください。

XAML の場合

次の手順に従います。

1. 次の XAML を <c1:C1Maps> タグと </c1:C1Maps> タグの間に追加します。

マークアップ

```
<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. プロジェクトを実行します。

コードの場合

1. XAML ビューで、x:Name="C1Maps1" を <c1:C1Maps> タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. コードビューに切り替えて、次の名前空間をインポートします。

Visual Basic

```
Imports C1.Xaml.Maps
```

C#

```
using C1.Xaml.Maps;
```

3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic コードの書き方

```
Visual Basic
' レイヤを作成してマップに追加します
Dim vl As C1VectorLayer = New C1VectorLayer()
C1Maps1.Layers.Add(vl)

'ベクタープレースマークを作成してレイヤに追加します
Dim vp1 As C1VectorPlacemark = New C1VectorPlacemark()
vl.Children.Add(vp1)

' プレースマークを一連の地理座標に設定します
vp1.GeoPoint = New Point(-80.107008, 42.16389)

' プレースマークのラベルとプロパティを設定します
vp1.Label = "Erie, PA"
vp1.FontSize = 12
vp1.Foreground = New SolidColorBrush(Colors.Red)
vp1.LabelPosition = LabelPosition.Center
```

C# コードの書き方

```
C#
// レイヤを作成してマップに追加します
C1VectorLayer vl = new C1VectorLayer();
C1Maps1.Layers.Add(vl);

//ベクタープレースマークを作成してレイヤに追加します
C1VectorPlacemark vp1 = new C1VectorPlacemark();
vl.Children.Add(vp1);

// プレースマークを一連の地理座標に設定します
vp1.GeoPoint = new Point(-80.107008, 42.16389);

// プレースマークのラベルとプロパティを設定します
vp1.Label = "Erie, PA";
vp1.FontSize = 12;
vp1.Foreground = new SolidColorBrush(Colors.Red);
vp1.LabelPosition = LabelPosition.Center;
```

4. プロジェクトを実行します。

このトピックの作業結果

次の図は、米国 Pennsylvania 州 Erie のラベルが付いた地理座標を含む C1Maps コントロールを示しています。

要素の表示/非表示

現在のマップスケールに応じて要素の表示/非表示を制御するためのプロパティがいくつかあります。たとえば、ズームインしたときには詳細な内容を表示し、ズームアウトしたときには詳細を隠すことができます。

全体的な制御は、要素が表示されるようになる最小比例画面サイズを **C1VectorLayer.MinSize** プロパティで指定して行います。

C1VectorPlacemark ラベルの表示/非表示を制御するために、特別なプロパティがあります。**C1VectorLayer.LabelVisibilty** には、次の値を指定できます。

- **Hide** - ラベルは不可視になり、ツールチップとして表示されます。
- **AutoHide** - 重なったラベルは非表示になります。
- **Visible** - すべてのラベルが表示されます。

さらに、各ベクター要素には **LOD** プロパティに格納される独自の表示/非表示設定があり、これがグローバル値より優先します。

LOD（Level of Details）構造体には次のプロパティが含まれます。

- **MinSize**、**MaxSize** - 要素の比例画面サイズの表示範囲を指定します。サイズがこの範囲に入らない場合、要素は非表示になります。
- **MinZoom**、**MaxZoom** - 代わりに、要素を表示するマップスケール（C1Maps.Zoom プロパティ）の範囲を指定します。

KML インポート/エクスポート

KML は、地理的な表示と注釈に使用される XML ベースの言語で、最初は Google Earth 用に作成されました。

KML インポートは、**KmlReader** クラスによって実行されます。このクラスには、用意された KML ソース（文字列またはストリーム）からベクターオブジェクトのコレクションを作成するための静的メソッドがあります。このコレクションは、**C1VectorLayer** に簡単に追加できます。インポートされるオブジェクトの **DataContext** は、KML ソースの対応する **XElement** に設定され、インポート中に元の要素を使ってカスタム操作を実行できます。

インポートには次の制限があります。

- KML プレースマーク要素のみがサポートされています。
- 内側多角形はサポートされていません。

- アイコンはサポートされていません。
- 外部リンクはサポートされていません。

KML エクスポートは、**KmlWriter** クラスによって実行されます。このクラスには、用意されたストリームにベクターオブジェクトのコレクションを KML フォーマットで書き込むための静的メソッドがあります。

KmlWriter.Write() メソッドには saveElementCallback パラメータがあり、これを使用して、エクスポート中にカスタム操作を実行できます。このメソッドは、KML ストリームに保存される要素ごとに呼び出されます。たとえば、コールバックメソッドを使用して、KML カスタムデータを要素に追加できます。

エクスポートには次の制限があります。

- **C1VectorPlacemark.Geometry** は KML ストリームに保存されません。

メモ: ShapeReader クラスを使用して ShapeFile および DBF ファイルをインポートできます。

データ連結

C1VectorLayer には、データ連結をサポートする2つのプロパティがあります。

- **ItemsSource** - ソースオブジェクトのコレクションを指定します。
- **ItemTemplate** - レイヤ上の各オブジェクトの外観を指定します。項目テンプレートでは、**C1VectorItemBase** を継承するクラスを定義する必要があります。

データ連結の例

City オブジェクトのコレクションがあるとします。

C#

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

このテンプレートは、**City** クラスから **C1VectorPlacemark** を作成する方法を定義します。

XAML マークアップ

```
<C1:C1Maps x:Name="maps" Foreground="LightGreen">
  <C1:C1Maps.Resources>
     <!-- 項目テンプレート -->
       <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>
```

最後に、データソースとして実際のコレクションを使用する必要があります。

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;
```

ズームレベル

C1Maps コントロールで、マップのズームレベルを指定できます。特定のマップポイントを設定する場合、それらのポイントでマップをズームしたいときがあります。**MaxZoom** プロパティと **MinZoom** プロパティで、ロード時のズームレベルを設定でき、また、マップにどの程度接近してズームできるかを制限できます。

メモ: **MaxZoom** プロパティに設定できる最大値は、**20** です。**MinZoom** プロパティに設定できる最小値は、**0** です。

この2つのプロパティは、XAML マークアップまたはコードを使用して簡単に設定できます。

XAML の場合

次のように、**<c1:C1Maps>** タグを編集します。

XAML マークアップ

```
<c1:C1Maps x:Name="maps" MinZoom="3" MaxZoom="15"/>
```

アプリケーションの実行時点では、ズームレベル "3" で表示されます。

コードの場合

次のコードを **InitializeComponent()** メソッドに追加します。

Visual Basic

```
maps.MaxZoom = 15
maps.MinZoom = 5
```

C#

```
maps.MaxZoom = 15;
maps.MinZoom = 5;
```

マップ範囲

範囲を制限することにより、C1Maps コントロール上の領域または州を強調表示するのは単純であり、次の4つのプロパティを設定することにより実行できます。

- **MaxLong** と **MinLong**
  このプロパティのペアは、経度によって指定される左および右側のマップ境界を制御します。
- **MaxLat** と **MinLat**
  このプロパティのペアは、緯度によって指定される上および下側のマップ境界を制御します。

これらのプロパティは、XAML マークアップ、コード、または設計時に[プロパティ]ウィンドウで設定できます。次の例では、設定された境界によってカリフォルニアが強調表示されます。

XAML の場合

次のように、**<c1:C1Maps/>** タグを編集します。これで、緯度値にカリフォルニア州の上と下の境界が設定されます。経度値には、カリフォルニア州の境界よりもわずかに外側が設定されます。

XAML マークアップ

```
<C1:C1Maps x:Name="maps" MaxLat="44" MinLat="32" Center="-121.224,37.8897" MaxLong="-112" MinLong="-126" MinZoom="5"/>
```

上のサンプルでは、Center プロパティと **MinZoomValue** プロパティも設定されています。

- **Center** プロパティを使用すると、特定の都市または領域が地図の中心になります。上のサンプルでは、カリフォルニア州のストックトンが中心になっています。
- **MinZoom** プロパティは、マップをロードする際の倍率を設定します。上のサンプルでは5に設定されており、マップが倍率5でズームされます。

コードの場合

次のコードは、マップ境界、マップの中心、およびロード時のズームレベルを設定します。これを **InitializeComponent()** メソッドに追加します。

C#

```
maps.MaxLat = 44;
maps.MinLat = 32;
maps.MaxLong = -112;
maps.MinLong = -126;
maps.Center = new Point(-121.224, 37.8897);
maps.MinZoom = 5;
```

参考のために、次の画像を作成するためのすべてのコードを示します。コードをコピーするには、名前空間宣言内の **YourProjectName** を自分のプロジェクトの名前空間に置き換えてください。

C#

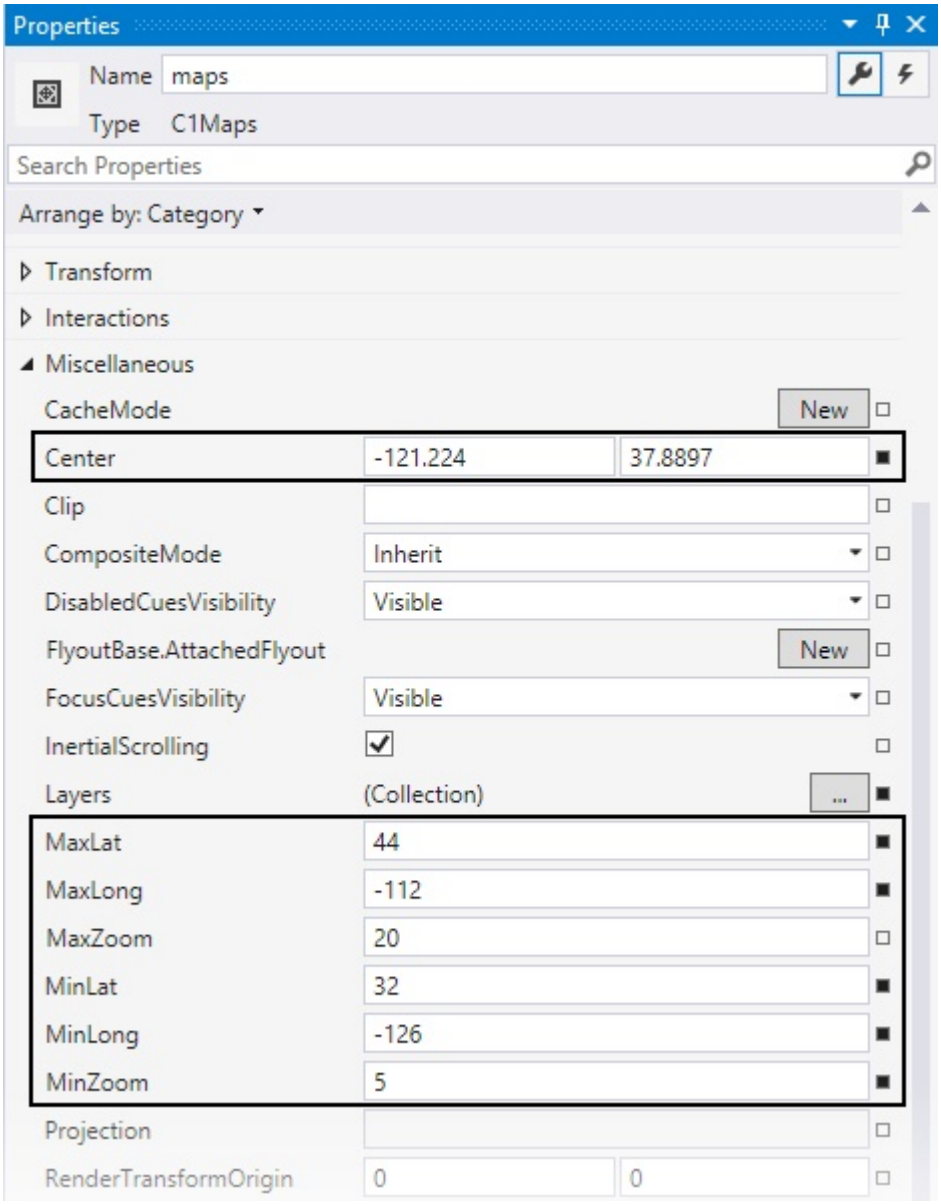
```
namespace YourProjectName
{
public class City
{
public Point LongLat { get; set; }
public string Name { get; set; }
}
```

```
public sealed partial class MainPage : Page
{
public MainPage()
{
this.InitializeComponent();

maps.MaxLat = 44;
maps.MinLat = 32;
maps.MaxLong = -112;
maps.MinLong = -126;
maps.Center = new Point(-121.224, 37.8897);
maps.MinZoom = 5;

City[] cities = new City[]
{
new City(){LongLat= new Point(-121.22361, 37.88972), Name="Stockton"},
new City(){LongLat= new Point(-117.1625, 32.7150), Name="San Diego"},
new City(){LongLat= new Point(-124.2017, 41.7558), Name="Crescent City"},
};
maps.DataContext = cities;
}
}
}
```

設計時の場合

[プロパティ]ウィンドウで、最小および最大の緯度値と経度値、**Center** プロパティ、および **MinZoomValue** プロパティを設定できます。

[プロパティ]ウィンドウの画像では、次のマップ画像を作成するためのプロパティが強調表示されています。アプリケーションでこれらのプロパティを設定するには、プロパティの横のテキストボックスをクリックし、適切な数値を入力します。



上の例は、次の図のようなマップになります。

メモ: 上の画像で、都市は、クイックスタート内のコードに類似するコードを使用してマークされていました。このトピックのコードはすべて、**[コードの場合]**タブにあります。次の XAML マークアップを、**<c1:C1Maps> <c1:C1Maps/>** タグの間に追加し、都市のコレクションの形式を設定して **C1Maps** コントロールに連結する必要があることに注意してください。

XAML マークアップ

```
<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}"/>
```

チュートリアル

このチュートリアルは、読者が Visual Studio .NET のプログラミングに精通していることを前提としています。ただし、**Maps for UWP** については、手順を追って説明されており、予備知識は特に必要ありません。このセクションに示される手順に従って作業を進めるだけで、**Maps for UWP** の機能を具体的に示すプロジェクトを作成できます。

チュートリアルプロジェクトを実行し、自分で変更してみてください。

クリックによるマップマーカーの追加

このチュートリアルでは、**C1Maps** を含む Windows ストアアプリケーションを Visual Studio で作成する方法について説明します。コードと別途コードファイルを追加すると、ユーザーが右クリックまたは右タップでマップマーカーを作成できるようになります。

手順1:アプリケーションの作成

この手順では、新しい Windows ストアアプリケーションを作成し、XAML ビューを設定します。

1. [**ファイル**]→[**新規作成**]→[**プロジェクト**]を選択し、[新しいプロジェクト]ダイアログボックスを開きます。
    1. 右側のペインで C# の下にある[**Windows ストア**]を選択します。
    2. 左側のペインで[**新しいアプリケーション (XAML)**]を選択します。
    3. アプリケーションの名前を入力し、[**OK**]をクリックします。新しい空の Windows ストアアプリケーションが開きます。
2. ソリューションエクスプローラーで、[**参照**]ファイルを右クリックし、リストから[**参照の追加**]を選択します。次のアセンブリ参照を参照して選択します。
    - C1.Xaml.dll
    - C1.Xaml.Maps.dll
    - C1.Xaml.Zip.dll
3. **MainPage.xaml** ファイルをダブルクリックして開きます。
4. ページ先頭の `<Page>` タグに次の名前空間宣言を追加します。
    - xmlns:C1="using:C1.Xaml.Maps"
    - xmlns:Xaml="using:C1.Xaml"

    タグは次のようになります。

    XAML マークアップ

    ```
    <Page
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:C1="using:C1.Xaml.Maps"
        xmlns:local="using:MapsTest5"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Xaml="using:C1.Xaml"
        x:Class="MapsTest5.MainPage"
        mc:Ignorable="d">
    ```

5. 次のマークアップを `<Grid> </Grid>` タグの間に挿入して、**C1Maps** コントロールと汎用の TextBlock コントロールをいくつかの書式設定と共に追加します。

    C#

    ```
    <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition />
    ```

```
</Grid.RowDefinitions>
<Border Margin="0,0,0,10">
<TextBlock TextWrapping="Wrap" Text="右クリックしてマークを追加し、
        マークをドラッグしてマークを移動し、マークをダブルクリックしてマークを削除します。" />
</Border>
<Border Grid.Row="1">
<Grid>
<C1:C1Maps x:Name="maps"/>
</Grid>
</Border>
```

この手順では、新しい Windows ストアアプリケーションを作成し、C1.Xaml 参照アセンブリへの参照を追加し、C1Maps コントロールと TextBlock コントロールを作成しました。

手順2:コードの追加

この手順では、**MainPage.xaml.cs** ファイルにコードを追加します。

1. **MainPage.xaml** ページを右クリックし、リストから[**コードの表示**]を選択します。**MainPage.xaml.cs** が開きます。
2. 次のコードをクラス宣言に追加します。

C#

```
C1VectorLayer vl;
Random rnd = new Random();
C1VectorPlacemark current = null;
Point offset = new Point();
```

3. MainPage()コンストラクタ内の**InitializeComponent()** メソッドに次のコードを追加します。このコードは、マップソースを変更し、C1VectorLayer を追加し、ランダムな座標を作成します。

C#

```
maps.Source = new VirtualEarthHybridSource();
            vl = new C1VectorLayer();
            maps.Layers.Add(vl);
            maps.RightTapped += maps_RightTapped;
            for (int i = 0; i < 10; i++)
            {
                // ランダムな座標を作成します
                Point pt = new Point(-80 + rnd.Next(160), -80 + rnd.Next(160));
                AddMark(pt);
            }       }
```

4. 次のコードは、MainPage()コンストラクタの閉じ中かっこの後に挿入する必要があります。このコードは、RightTapped イベントハンドラと AddMark() メソッドを追加します。

C# コードの書き方

C#

```
void maps_RightTapped(object sender, RightTappedRoutedEventArgs e)
{
AddMark(maps.ScreenToGeographic(e.GetPosition(maps)));
}
void AddMark(Point pt)
{
```

```
Color clr = Utils.GetRandomColor(128, 192);
C1VectorPlacemark mark = new C1VectorPlacemark()
{
GeoPoint = pt,
Label = new TextBlock()
{
RenderTransform = new TranslateTransform() { Y = -5 },
IsHitTestVisible = false,
Text = (vl.Children.Count + 1).ToString()
},
LabelPosition = LabelPosition.Top,
Geometry = Utils.CreateBaloon(),
Stroke = new SolidColorBrush(Colors.DarkGray),
Fill = new SolidColorBrush(clr),
};
mark.PointerPressed += mark_PointerPressed;
maps.PointerMoved += mark_PointerMoved;
mark.PointerReleased += mark_PointerReleased;
vl.Children.Add(mark);
mark.DoubleTapped += mark_DoubleTapped;
}
```

5. 最後に追加するコードセクションでは、C1Maps コントロールのいくつかの Pointer イベントと、マーカーの DoubleTapped イベントを処理します。

C# コードの書き方

C#

```
void mark_PointerMoved(object sender, PointerRoutedEventArgs e)
{
if (current == null)
{
return;
}
var cur = e.GetCurrentPoint(maps).Position;
cur = new Point(cur.X - offset.X,
cur.Y - offset.Y);
current.GeoPoint = maps.ScreenToGeographic(cur);
e.Handled = true;
}
void mark_PointerPressed(object sender, PointerRoutedEventArgs e)
{
offset = e.GetCurrentPoint(sender as C1VectorPlacemark).Position;
current = sender as C1VectorPlacemark;
e.Handled = true;
}
void mark_PointerReleased(object sender, PointerRoutedEventArgs e)
{
current = null;
e.Handled = true;
}
void mark_DoubleTapped(object sender, DoubleTappedRoutedEventArgs e)
{
```

```
e.Handled = true;
vl.Children.Remove((C1VectorPlacemark)sender);
}
```

この手順では、タップイベントなどのイベントを処理するコードを追加しました。また、マップソースを変更し、初期マップマーカーの座標をランダムに作成しました。

手順3:コードファイルの追加

この手順では、マーカーバルーンの作成や色付けを処理するコードファイルを追加します。

1. アプリケーション名を右クリックし、リストから[**追加**]→[**新しい項目**]を選択します。
2. 左側ペインで[**コード**]選択したら、左側ペインで**コードファイル**を選択します。
3. 新しいコードファイルに **Utils.cs** という名前を付け、[**OK**]をクリックすると、**Utils.cs** が開きます。
4. 次の名前空間をインポートします。

C# コードの書き方

C#

```
using C1.Xaml.Maps;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Windows.Foundation;
using Windows.UI;
using Windows.UI.Xaml.Media;
using System.Reflection;
```

5. 名前空間宣言を次のように追加します。
6. 名前空間宣言の下に、マーカーの作成を処理し、マーカーをランダムな色で表示する次のコードを追加します。

C# コードの書き方

C#

```
namespace YourProjectNameHere
                {
```

7. 名前空間宣言の下に、マーカーの作成を処理し、マーカーをランダムな色で表示する次のコードを追加します。

C# コードの書き方

C#

```
public class Utils
{
public static Geometry CreateBaloon()
{
PathGeometry pg = new PathGeometry();
pg.Transform = new TranslateTransform() { X = -10, Y = -24.14 };
PathFigure pf = new PathFigure() { StartPoint = new Point(10, 24.14),
IsFilled = true, IsClosed = true };
pf.Segments.Add(new ArcSegment() { SweepDirection = SweepDirection.
Counterclockwise, Point = new Point(5, 19.14), RotationAngle = 45,
Size = new Size(10, 10) });
pf.Segments.Add(new ArcSegment() { SweepDirection = SweepDirection.Clockwise,
Point = new Point(15, 19.14), RotationAngle = 270, Size = new Size(10, 10),
IsLargeArc = true });
```

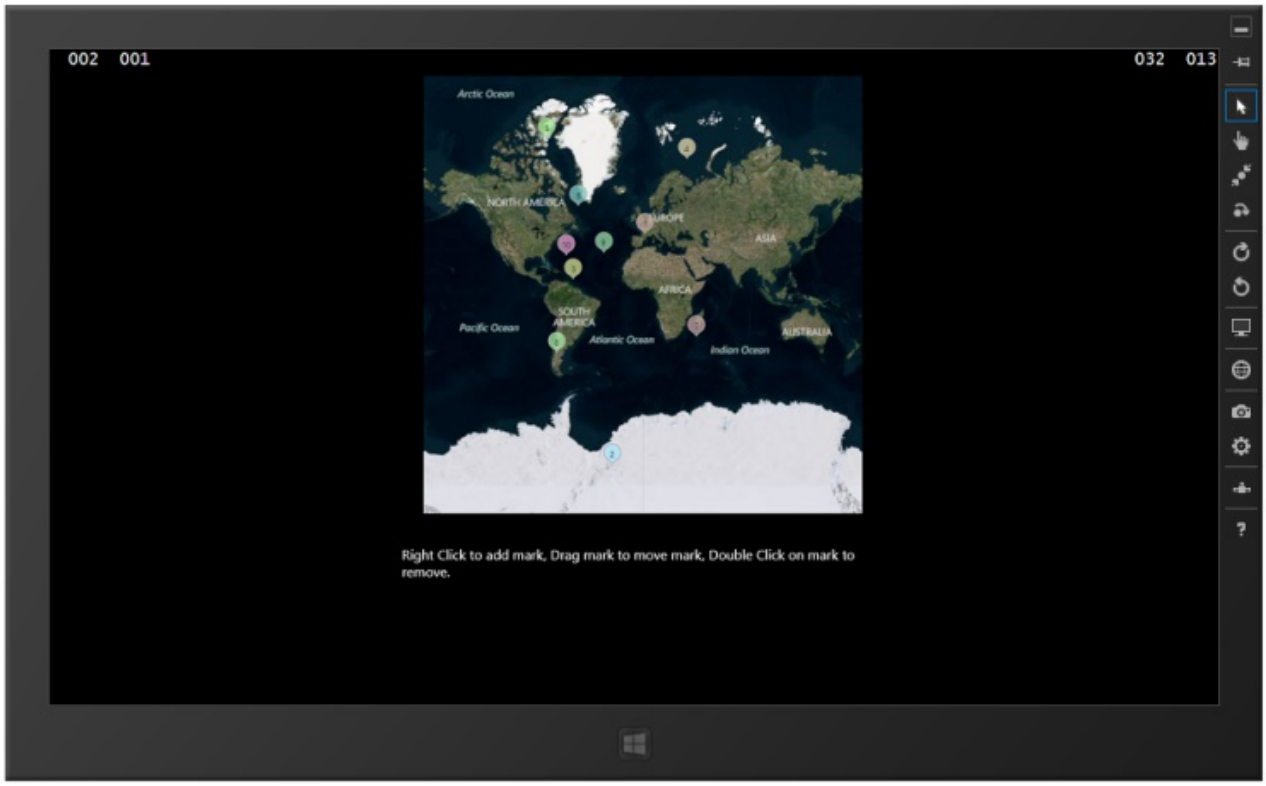
```
pf.Segments.Add(new ArcSegment() { SweepDirection = SweepDirection.Counterclockwise,
Point = new Point(10, 24.14), RotationAngle = 45, Size = new Size(10, 10) });
pg.Figures.Add(pf);
return pg;
}

static Random rnd = new Random();
public static Color GetRandomColor(byte a)
{
return Color.FromArgb(a, (byte)rnd.Next(255), (byte)rnd.Next(255), (byte)rnd.Next(255));
}
public static Color GetRandomColor(byte min, byte a)
{
return Color.FromArgb(a, (byte)(min + rnd.Next(255 - min)),
(byte)(min + rnd.Next(255 - min)), (byte)(min + rnd.Next(255 - min)));
}
}
        }
```

この手順では、マーカーバルーンの作成とマーカーのランダムな色付けを処理するコードファイルを追加しました。次の手順では、このアプリケーションを実行します。

手順4:アプリケーションの実行

この手順では、アプリケーションを実行します。

1. [F5]キーを押すかデバッグを開始して、アプリケーションを実行します。次の図のように表示されます。マップの初期表示では、マーカーが 10 個表示されます。



2. マップを右タップするか、右クリックすると、連番付きのマーカーが追加されます。

おめでとうございます。これで、マップマーカーの追加チュートリアルは終了です。このチュートリアルでは、Windows ストアアプリケーションを作成し、C1Maps アプリケーションを作成するためのマークアップとコードを追加し、コードファイルを追加しました。

C1VectorPolylineのルートをマークする

C1VectorPolyline クラスについてはじめては「ベクターオブジェクト」トピックに紹介しました。
そのトピックでは、マップで3つのポイントを連結している折れ線を作成します。**C1VectorPolyline**を使用して、より複雑なルートマップを作成できます。

このチュートリアルでは、ルートにある主な都市や停をマークしてシベリア横断鉄道のルートを示しているプロジェクトを作成します。

手順1:アプリケーションの作成

この手順では、アプリケーションを作成し、そのアプリケーションにいくつかのフォルダを追加します。

この手順では、新しい Windows ストアアプリケーションを作成し、XAML ビューを設定します。

1. **[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択し、**[新しいプロジェクト]**ダイアログボックスを開きます。
    1. 右側のペインで C# の下にある[Windows ストア]を選択します。
    2. 左側のペインで**[新しいアプリケーション (XAML)]**を選択します。
    3. アプリケーションの名前を入力し、[**OK**]をクリックします。新しい空の Windows ストアアプリケーションが開きます。

2. ソリューションエクスプローラーで、**[参照]**ファイルを右クリックし、リストから**[参照の追加]**を選択します。次のアセンブリ参照を参照して選択します。
    - C1.Xaml.dll
    - C1.Xaml.DX.dll
    - C1.Xaml.Maps.dll
    - C1.Xaml.Zip.dll

3. **MainPage.xaml** ファイルをダブルクリックして開きます。

4. Visual Studio ツールボックスで C1Maps コントロールを見つけてダブルクリックし、このコントロールをアプリケーションに追加します。
5. **<Maps: C1Maps/>** タグを次のように編集します。

XAML

```
<Maps:C1Maps Name="maps" Foreground="White" ></Maps:C1Maps>
```

6. 次に、最小および最大の緯度値と経度値を設定して、マップの境界を設定します。マークアップを次のように編集します。

XAML

```
<Maps:C1Maps Name="maps" Foreground="White" MaxLatValue="70" MinLat="35"
MaxLong="146" MinLongValue="25" ></Maps:C1Maps>
```

7. ここで、**<Maps:C1Maps>** タグを編集して中心を設定します。このプロジェクトでは、Center プロパティにノボシビルスクの緯度値と経度値を設定します。

XAML

```
<Maps:C1Maps Name="maps" Foreground="White" MaxLat="70" MinLat="35"
MaxLong="146" MinLong="25" Center="82.9333, 55.0167" ></Maps:C1Maps>
```

8. 次に、MinZoom を設定します。

XAML

```
<Maps:C1Maps Name="maps" Foreground="White" MaxLat="70" MinLat="35"
MaxLong="146" MinLong="25" Center="82.9333, 55.0167" MinZoom="3"> </Maps:C1Maps>
```

9. カーソルを **<Maps:C1Maps> </Maps:C1Maps>** タグの間に置きます。次のマップリソースと C1VectorLayer を追加します。

XAML

```
<Maps:C1Maps.Resources>
<!--Item template -->
<DataTemplate x:Key="templPts">
<Maps:C1VectorPlacemark GeoPoint="{Binding Path=LongLat}" Fill="LightGreen"
Stroke="DarkGreen" Label="{Binding Path=Name}" LabelPosition="Right"
FontSize="14">
<Maps:C1VectorPlacemark.Geometry>
<EllipseGeometry RadiusX="2"
RadiusY="2" />
</Maps:C1VectorPlacemark.Geometry>
</Maps:C1VectorPlacemark>
</DataTemplate>
</Maps:C1Maps.Resources>
<Maps:C1VectorLayer ItemsSource="{Binding}"
        ItemTemplate="{StaticResource templPts}"/>
```

この手順では、アプリケーションを作成し、適切な参照を追加し、XAML マークアップを追加して、**C1Maps** コントロール、マップリソース、および **C1VectorLayer** を作成しました。

手順2:コードの追加

この手順では、**C1VectorPolyline** とマップ上の都市の両方を作成するコードを追加します。

1. コードビューに切り替えて、次の名前空間をインポートします。

C#

```
using C1.Xaml.Maps;
```

2. プロジェクトの名前空間宣言のすぐ下に、次のクラスを追加します。

C#

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

3. **InitializeComponent()** メソッドの下で、**C1VectorPolyline** を入れる C1VectorLayer の作成を開始します。

C#

```
// レイヤを作成してマップに追加します
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
maps.Layers.Add(C1VectorLayer1);
```

4. 次に、**C1VectorPolyline** のポイントを作成します。

C#

```
// 初期経路
Point[] pts = new Point[]
{
    new Point(37.6167, 55.7500),
    new Point(40.4167, 56.1333),
    new Point(44.0075, 56.3269),
    new Point(49.6500, 58.6000),
    new Point(56.3167, 58.0000),
    new Point(60.5833, 56.8333),
    new Point(65.5333, 57.1500),
    new Point(73.3667, 54.9833),
    new Point(82.9333, 55.0167),
    new Point(93.0667, 56.0167),
    new Point(98.0167, 55.9500),
    new Point(104.2958, 52.3122),
    new Point(103.6833, 51.7176),
    new Point(103.7500, 51.6333),
    new Point(104.0000, 51.3020),
    new Point(105.8667, 51.7176),
    new Point(107.6000, 51.8333),
    new Point(113.4667, 52.0500),
    new Point(123.9333, 53.9833),
    new Point(128.4667, 50.9167),
    new Point(132.9011, 48.8014),
    new Point(135.0667, 48.4833),
    new Point(131.9667, 43.8000),
    new Point(131.9000, 43.1333),
};
```

5. ポイントのコレクションを作成し、作成済みのポイントをその中に含めます。

C#

```
// コレクションを作成し、データを設定します
PointCollection pcoll = new PointCollection();
foreach (Point pt in pts)
    pcoll.Add(pt);
```

6. 折れ線をベクターレイヤに追加し、ポイントのコレクションをベクターレイヤに割り当て、外観をカスタマイズします。

C#

```
// 折れ線を作成して、ベクターレイヤに子として追加します
C1VectorPolyline C1VectorPolyline1 = new C1VectorPolyline();
C1VectorLayer1.Children.Add(C1VectorPolyline1);
// ポイント
C1VectorPolyline1.Points = pcoll;
// 外観
C1VectorPolyline1.Stroke = new SolidColorBrush(Colors.Red);
C1VectorPolyline1.StrokeThickness = 1;
```

7. 最後に、都市のリストを作成して、**DataContext** で **C1Maps** コントロールに追加します。これらの都市は、シベリア横断鉄道の主な停車駅を示しています。

C#

```
      City[] cities = new City[]
{
    new City(){LongLat= new Point(37.6167, 55.7500), Name="Moscow"},
    new City(){LongLat= new Point(40.4167, 56.1333), Name="Vladimir"},
    new City(){LongLat= new Point(44.0075, 56.3269), Name="Nizhny Novgorod"},
    new City(){LongLat= new Point(49.65, 58.6), Name="Kirov"},
    new City(){LongLat= new Point(56.3167, 58.000), Name="Perm"},
    new City(){LongLat= new Point(60.5833, 56.8333), Name="Yekaterinburg"},
    new City(){LongLat= new Point(65.5333, 57.1500), Name="Tyumen"},
    new City(){LongLat= new Point(73.3667, 54.9833), Name="Omsk"},
    new City(){LongLat= new Point(82.9333, 55.0167), Name="Novosibirsk"},
    new City(){LongLat= new Point(93.0667, 56.0167), Name="Krasnoyarsk"},
    new City(){LongLat= new Point(98.0167, 55.9500), Name="Tayshet"},
    new City(){LongLat= new Point(104.2958, 52.3122), Name="Irkutsk"},
    new City(){LongLat= new Point(107.6000, 51.8333), Name="Ulan Ude"},
    new City(){LongLat= new Point(113.4667, 52.05), Name="Chita"},
    new City(){LongLat= new Point(132.9011, 48.8014), Name="Birobidzhan"},
    new City(){LongLat= new Point(135.0667, 48.4833), Name="Khabarovsk"},
    new City(){LongLat= new Point(131.9667, 43.8000), Name="Ussuriysk"},
    new City(){LongLat= new Point(131.9, 43.1333), Name="Vladivostok"},
};
      maps.DataContext = cities;
  }
```

この手順では、アプリケーションのコードを作成しました。このコードは、**C1VectorLayer** とポイントのコレクションを作成し、ポイントをコレクションに追加し、さらにポイントを **C1VectorPolyline** に割り当てました。次に、**C1VectorPolyline** の外観を調整し、マップ上でマークされる都市のリストを追加しました。

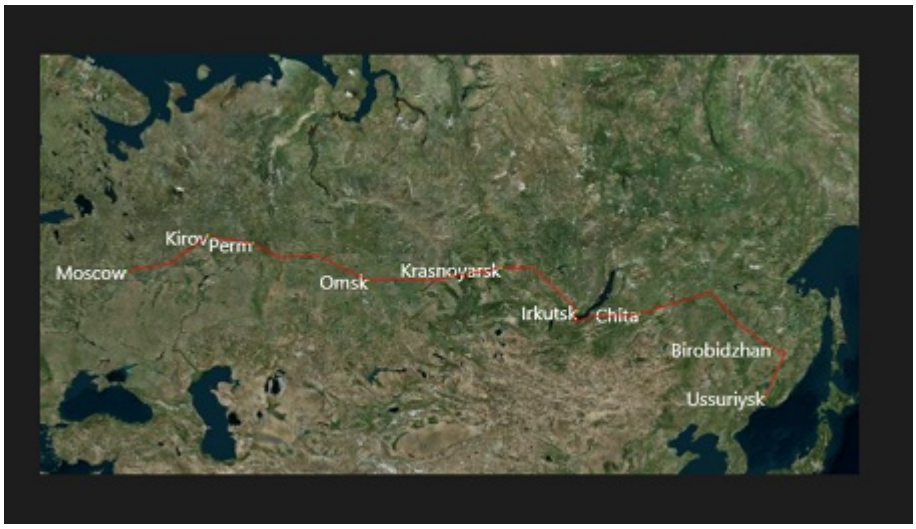
次の手順では、このアプリケーションを実行します。

手順3:アプリケーションの実行

この手順ではアプリケーションを実行します。

1. [F5]キーまたは [デバッグ開始] を押して、アプリケーションを実行します。アプリケーションは次の図のようになります。

   次の図は、大まかにシベリア横断鉄道に従う折れ線を示しています。正確なルートを示すマップを表示するには、さらに多くのポイントが必要です。マップ上の折れ線は主な都市を通り、メインルートに従っています。

   コレクションのすべての都市がマップに表示されているわけではないことに注意してください。

2. タップするか、クリックするか、マウスホイールを使用して、マップをウスリースクにズームインします。マップ上の最終地点、ウラジオストクを確認できるはずです。

おめでとうございます。

このチュートリアルでは、アプリケーションを作成し、C1Maps コントロールを追加し、XAML マークアップを編集してカスタマイズし、C1VectorPolyline を作成するコードを追加し、さらにシベリア横断鉄道の主な地点を示すために都市のコレクションを追加しました。

同じ形式を使用して、任意のルートのコースを示すことができます。