Maps for WPF/Silverlight

2018.04.11 更新

グレープシティ株式会社

目次

製品の概要

Maps for WPF/Silverlight 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。

> **メモ:** 説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則としてWPF版のリファレンスページを参照します。Silverlight版については、目次から同名のメンバーを参照してください。

主な特長

Maps for WPF/Silverlight 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 WPF/Silverlight** 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。

  a. 行政界（国境、県境など）

  b. 地理情報（路線図、航路図の表示など）

  c. 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 WPF/Silverlight 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。

- **ズーム、パン、マップ座標**

  Maps for WPF/Silverlight では、マウスまたはキーボードを使ってズームや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。

クイックスタート

このクイックスタートガイドは、**Maps for WPF/Silverlight** 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、Expression Blend で **C1Maps** コントロールを含む新しいプロジェクトを作成します。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 Expression Blend で **C1Maps** コントロールを使用する WPF/Silverlight 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。
次の手順に従います。

1. Expression Blend で、[**ファイル**]→[**新しいプロジェクト**]を選択します。
2. [**新しいプロジェクト**]ダイアログボックスで、左ペインからプロジェクトの種類として[WPF/Silverlight]を選択し、右ペインから[**WPF/Silverlight アプリケーション + Web サイト**]を選択します。
3. プロジェクトの**[名前]**と[**場所**]を入力し、ドロップダウンボックスで[**言語**]を選択し、[**OK**]をクリックします。Blend によって作成された新しいアプリケーションが開き、デザインビューに **MainPage.xaml** ファイルが表示されます。
4. 次の手順に従って、**C1Maps** コントロールをプロジェクトに追加します。
   a. **[プロジェクト]→[参照の追加]**を選択します。
   b. **Maps for WPF/Silverlight** でインストールされている **C1.WPF.Maps.dll** または **C1.Silverlight.Maps.dll** アセンブリを見つけて参照します。

      > **注意: デフォルトでは、C1.WPF.Maps.dll** または **C1.Silverlight.Maps.dll** ファイルは「C:\Program Files\ComponentOne\WPF\Bin または C:\Program Files\ComponentOne\Silverlight\Bin」にインストールしています。

   c. **C1.WPF.dll** を選択し、**[開く]**をクリックします。これで、プロジェクトにリファレンスが追加されます。
5. 次の手順に従って、**C1Maps** コントロールをプロジェクトに追加します。
   a. メニューから**[ウィンドウ]→[アセット]**を選択して**[アセット]**タブを開きます。
   b. **[アセット]**タブで、検索バーに「C1Maps」と入力します。
   c. **C1Maps** コントロールのアイコンが表示されます。
   d. [**C1Maps**]アイコンをダブルクリックしてコントロールをプロジェクトに追加します。
6. [**オブジェクトとタイムライン**]パネルで[**C1Maps**]を選択し、[**プロパティ**]パネルで次のプロパティを設定します。
   a. **Name** プロパティを "C1Maps1"に設定します。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。
   b. **Width** プロパティを "405" に設定します。
   c. **Height** プロパティを "472" に設定します。
   d. **Zoom** プロパティを "2" に設定して、ズーム倍率を元の倍率の2倍に設定します。
   e. **Center**プロパティを "-65, -25" に設定して、南米のみがマップに表示されるようにします。

この手順では、Blend Silverlight プロジェクトを作成し、**C1Maps** コントロールを追加しました。また、**C1Maps** コントロールのいくつかのプロパティを設定しました。

手順 2：データソースへの連結

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

```
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; }
  }
```

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

```
Dim cities() As City =
New City() {
   New City(New Point(-58.40, -34.36),  "ブエノスアイレス"),
   New City(New Point(-47.92, -15.78), "ブラジリア"),
   New City(New Point(-70.39, -33.26), "サンティアゴ"),
   New City(New Point(-78.35, -0.15), "キト"),
   New City(New Point(-66.55, 10.30), "カラカス"),
   New City(New Point(-77.03, -12.03), "リマ"),
   New City(New Point(-57.40, -25.16), "アスンシオン"),
   New City(New Point(-74.05, 4.36), "ボゴタ"),
   New City(New Point(-68.09, -16.30), "ラパス"),
   New City(New Point(-58.10, 6.48), "ジョージタウン"),
   New City(New Point(-55.10, 5.50), "パラマリボ"),
   New City(New Point(-56.11, -34.53),"モンテビデオ")
   }
C1Maps.DataContext = cities
```

Visual Basic

```
City[] cities = new City[]
{
   new City(){ LongLat= new Point(-58.40, -34.36), Name="ブエノスアイレス"},
   new City(){ LongLat= new Point(-47.92, -15.78), Name="ブラジリア"},
   new City(){ LongLat= new Point(-70.39, -33.26), Name="サンティアゴ"},
   new City(){ LongLat= new Point(-78.35, -0.15), Name="キト"},
   new City(){ LongLat= new Point(-66.55, 10.30), Name="カラカス"},
   new City(){ LongLat= new Point(-56.11, -34.53), Name="モンテビデオ"},
   new City(){ LongLat= new Point(-77.03, -12.03), Name="リマ"},
   new City(){ LongLat= new Point(-57.40, -25.16), Name="アスンシオン"},
   new City(){ LongLat= new Point(-74.05, 4.36), Name="ボゴタ"},
   new City(){ LongLat= new Point(-68.09, -16.30), Name="ラパス"},
   new City(){ LongLat= new Point(-58.10, 6.48), Name="ジョージタウン"},
   new City(){ LongLat= new Point(-55.10, 5.50), Name="パラマリボ"},
};
C1Maps.DataContext = cities;
```

4. XAML ビューに切り替えて、開始タグと終了タグが含まれるように <c1:C1Maps>マークアップを変更します。次のようになります。

XAML

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

5. Foreground="Aqua" を <c1:C1Maps>タグに追加します。

6. 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

```
<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。

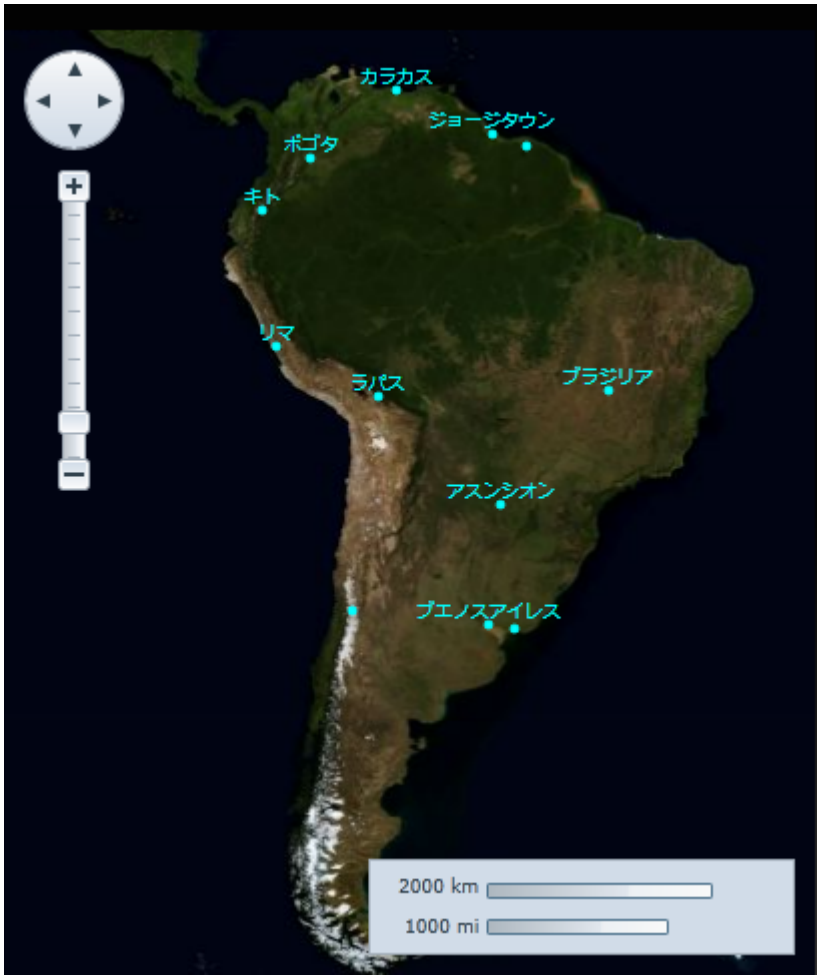
この手順では、データソースを作成し、それを**C1VectorPlacemark** のプロパティに連結しました。次の手順では、プログラムを実行して、このクイックスタートガイドで作成したプロジェクトの結果を表示します。

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 WPF/Silverlight 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。

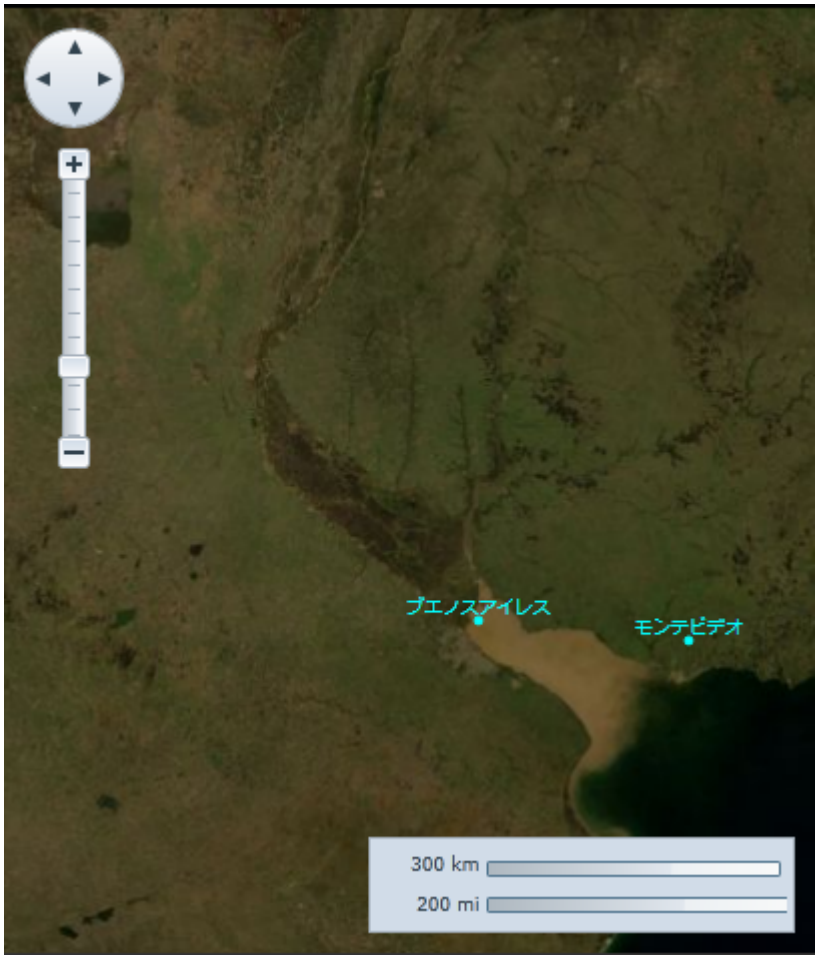
次の手順に従います。

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。



ブエノスアイレスの近くと ジョージタウン の近くにそれぞれ点があり、その2つの点の横には名前がないことを確認します。

2. **ブエノスアイレス**の近くをダブルクリックします。これを2回繰り返して、**モンテビデオ** というラベルがマップに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 WPF/Silverlight** クイックスタートは終了です。ヘルプファイルの「Maps の使い方」および「タスク別ヘルプ」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。

Maps の使い方

C1.WPF.Maps または **C1.Silverlight.Maps** アセンブリには、独自のカスタムデータや Bing Maps などのさまざまなソースから取得した豊富な地理情報を表示する **C1Maps** コントロールが含まれます。

C1Maps は、ズーム、パン、および画面と地理座標の間のマッピングをサポートします。また、マップに要素を重ねるためのレイヤもサポートされます。レイヤは項目の仮想化をサポートし、静的な要素のほか、地理的な位置に関連付けられた要素を表示することを可能にします。

次のトピックでは、**C1Maps** コントロールの基本的な使用方法について説明します。

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

HTTPS サポート

セキュリティ上の理由から、Microsoft Silverlight では、ゾーン間、ドメイン間、およびスキーム間の URL アクセスが制限されています。次の表に、これらの規則を示します。

	ダウンローダーオブジェクト	メディア、画像、ASX	XAML ファイル、フォントファイル	ストリーミングメディア
許可されるスキーム	HTTP、HTTPS	HTTP、HTTPS、FILE	HTTP、HTTPS、FILE	HTTP
スキーム間アクセス	不可	不可	不可	HTTPS からは不可
Web ドメイン間アクセス	不可	HTTPS を除く	不可	可
ゾーン間アクセス（Windows）	不可	不可	不可	不可
ゾーン間アクセス（Macintosh）	不可	可	不可	可
リダイレクトの許可	同じドメイン内（Firefox/Safari のみ）	同じドメイン内	同じドメイン内	不可

WPF HTTPS サポートの詳細については、MSDN の「**WPF URL アクセスポリシー**」（http://msdn.microsoft.com/ja-jp/library/bb820909.aspx）を参照してください。

> **メモ:** HTTPS では **C1Maps** コントロールを使用できます。ただし、画像タイルは WPF アプリケーションと同じドメインに格納されている必要があります。

マップの概念と主要なプロパティ

このセクションでは、基本的な **C1Maps** の概念と主要なプロパティについて説明します。

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 **BingMaps** 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

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

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

- Virtual Earth Road ソース

```
Visual Basic
map2.Source = New VirtualEarthRoadSource()
```

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

- Virtual Earth Hybrid ソース

```
Visual Basic
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、マウスでマップをドラッグしたり、左上隅に表示されるナビゲータコントロールを使用して、**Center** プロパティの値を変更できます。

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。ユーザーは、マウスホイールまたはコントロールの左側にあるズームコントロールを使用して、Zoom プロパティの値を変更できます。

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。

- **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。
- **C1MapToolsLayer** は、パンとズームのツールやスケールを表示するために使用される2番目のレイヤです。このレイヤは **C1Maps** のテンプレートに組み込まれているため、手作業で追加する必要がありません。
- **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。

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

```
<c1:C1Maps>
    <c1:C1Maps.Layers>
        <c1:C1MapItemsLayer>
            <Ellipse Width="20" Height="20" Fill="Red" c1maps:C1MapCanvas.LatLong="-
79.9247, 40.4587" c1maps: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。

C#

```
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。

```
<DataTemplate x:Name="itemTemplate">
    <StackPanel Orientation="Horizontal" c1maps:C1MapCanvas.LatLong="{Binding
LatLong}" c1maps:C1MapCanvas.Pinpoint="5, 5">
        <Ellipse Fill="Red" Width="10" Height="10" />
        <TextBlock Text="{Binding Name}" Foreground="White" />
    </StackPanel>
</DataTemplate>
```

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

```
<DataTemplate x:Key="itemTemplate">
    <StackPanel Orientation="Horizontal" c1maps:C1MapCanvas.LatLong="{Binding
LatLong}" c1maps: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。

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 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** – Polygon クラスに似ていますが、このオブジェクトは閉じた形状にする必要はありません。地理座標を使って折れ線が形成されます。用途の例として、道路や経路があります。タスク別ヘルプについては、「折れ線を追加する」を参照してください。
- **C1VectorPolygon** – Polyline クラスに似ていますが、これは多角形を描画します。多角形は、接続線で閉じた形状を形成します。地理座標を使って多角形が形成されます。用途の例として、境界や領域があります。タスク別ヘルプについては、「多角形を追加する」を参照してください。
- **C1VectorPlacemark** – 地理的な場所に関連付けられたオブジェクト。プレースマークは、ピクセル座標で座標が表現される、スケールに依存しないジオメトリと、オプションのラベル（任意の UIElement）を持ちます。用途の例として、マップ上のラベル、アイコン、マークがあります。タスク別ヘルプについては、「ラベルを追加する」を参照してください。

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（**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。

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

```
<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;
```

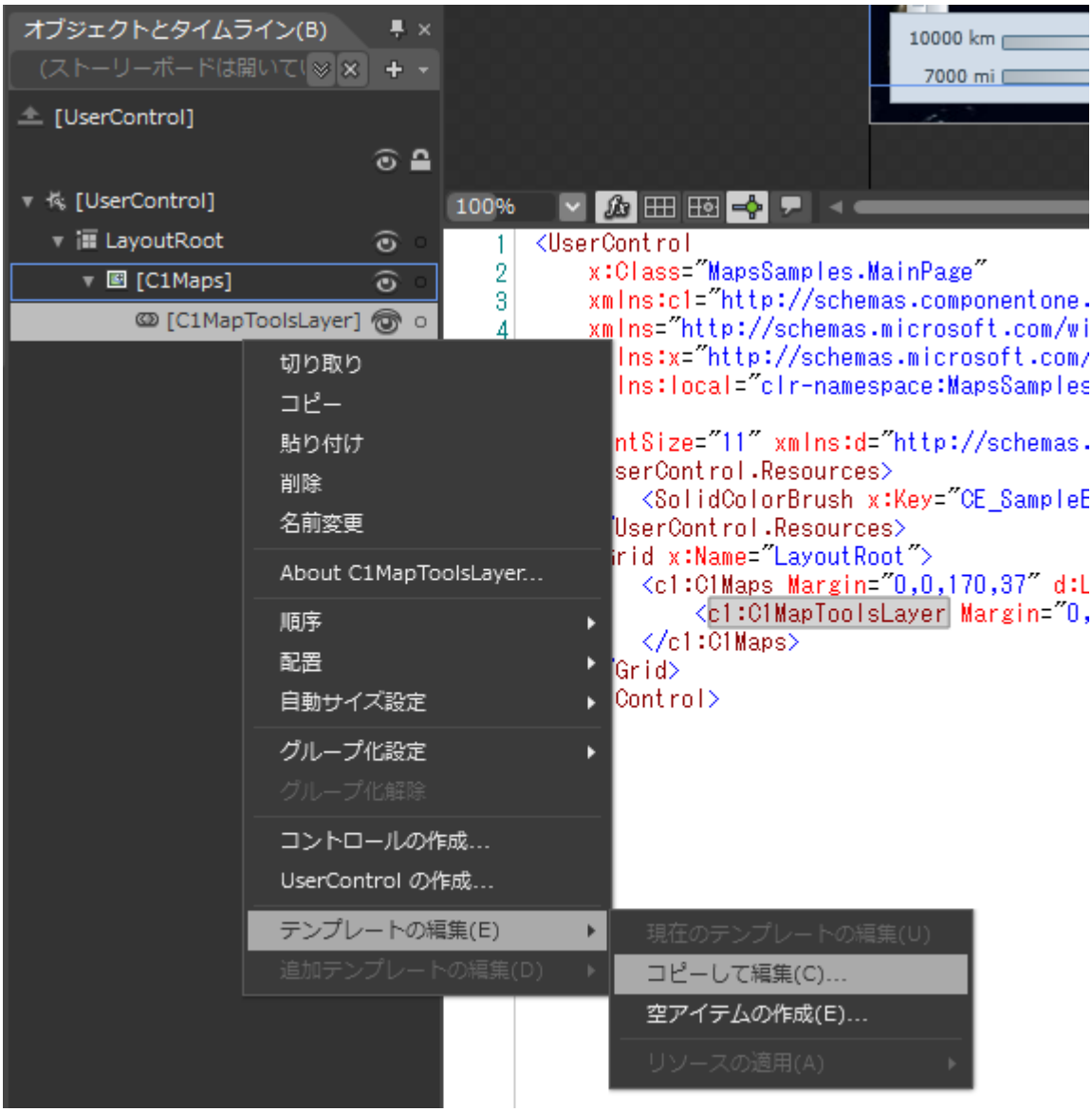
ツールのカスタマイズ

デフォルトでマップに表示されるパンツールとズームツールは、**C1MapToolsLayer** に実装されています。これは **C1Maps** のテンプレートに入っているため、Layers コレクションに追加する必要はありません。これらのツールをカスタマイズするには、**C1Maps.ShowTools** を False に設定してデフォルトのツールを非表示にしてから、独自の **C1MapToolsLayer** インスタンスを追加します。このための XAML を次に示します。

XAML

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

ツールのためにまったく別のレイヤを実装することもできますが、この例では、単に組み込みツールのテンプレートを変更しています。この XAML を Blend で編集するには、**ToolsLayer** を右クリックし、**[コントロールパーツの編集(テンプレート)]→[コピーして編集]**を選択します。



これで、Blend でテンプレートを編集すると、変更がマップに反映されます。

レイアウトと外観

以下のトピックでは、**C1Maps** コントロールのレイアウトと外観をカスタマイズする方法について詳しく説明します。組み込みのレイアウトオプションを使用して、グリッドやキャンバスなどのコントロールをパネル内でレイアウトできます。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、コントロールを書式設定およびレイアウトしたり、コントロールの操作をカスタマイズすることもできます。

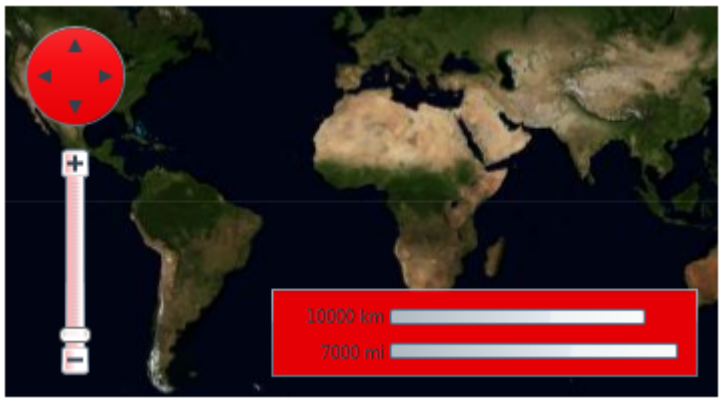
C1Maps の ClearStyle プロパティ

Maps for WPF/Silverlight は、コントロールのテンプレートを変更することなくコントロールの色を簡単に変更できる ComponentOne の新しい ClearStyle 技術をサポートします。色のプロパティをいくつか設定するだけで、グリッド全体のスタイルを簡単に設定できます。

次の表に、**C1Maps** コントロールのブラシのプロパティの概要を示します。

ブラシ	説明
背景	コントロールの背景のブラシを取得または設定します。
MouseOverBrush	マウスポインタが置かれたマップボタンを強調表示するために使用される System.Windows.Media.Brush を取得または設定します。
PressedBrush	クリックされたボタンを強調表示するために使用される System.Windows.Media.Brush を取得または設定します。

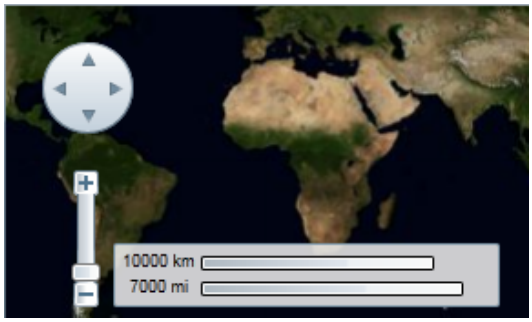
いくつかのプロパティを設定することで、C1Maps コントロールの外観を完全に変更できます。たとえば、Background は、マップのツールの背景色を設定します。Background プロパティを "##FFE40005" に設定すると、C1Maps コントロールは次のようになります。



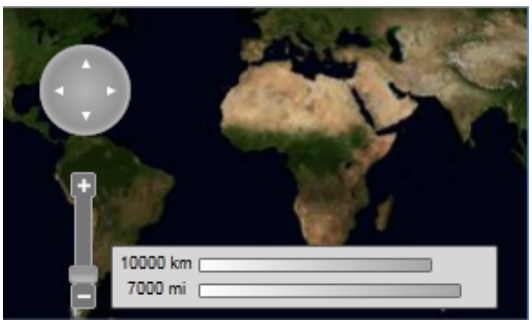
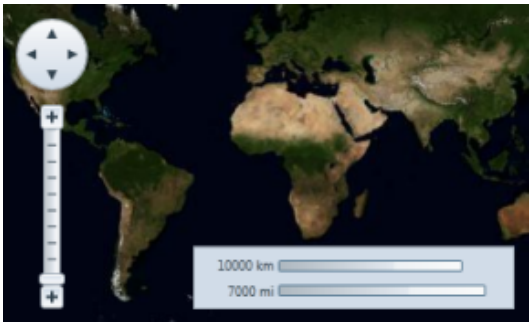
ComponentOne の ClearStyle 技術は、このように簡単に使用できます。

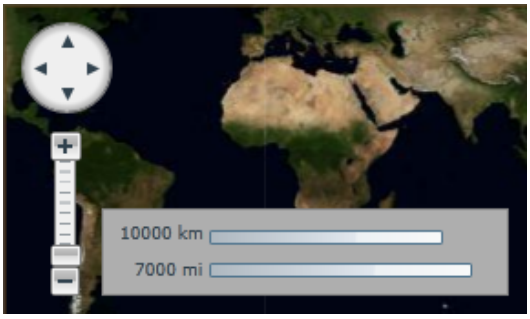
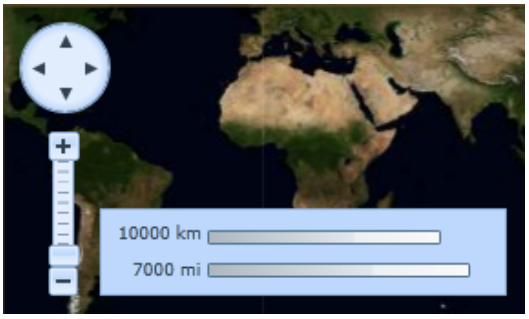
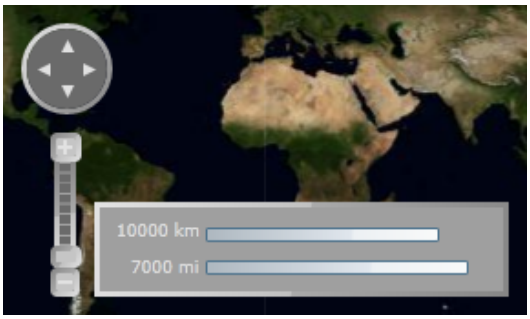
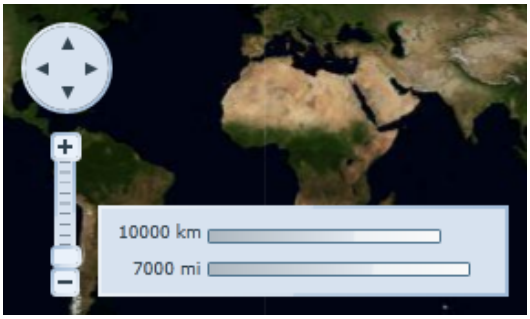
テーマ

Maps for WPF/Silverlight には、コントロールをカスタマイズできるようにいくつかのテーマが組み込まれています。まずページに C1Maps コントロールを追加すると、コントロールはデフォルトのテーマで表示されます。



これはコントロールのデフォルトの外観です。この外観を変更するには、組み込みテーマの1つを使用するか、独自のカスタムテーマを作成することができます。すべての組み込みテーマは WPF Toolkit のテーマに基づいています。次の組み込みテーマでは、選択したスタイルを表示するように、行が選択されています。

完全なテーマ名	外観
C1ThemeBureauBlack	
C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1Blue (WPF のみ)	

C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	

C1ThemeOffice2010Silver	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

要素のテーマを設定するには、**ApplyTheme** メソッドを使用します。まずプロジェクトにテーマアセンブリへの参照を追加して、次のコードのようにテーマを設定します。

Visual Basic

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

    ' ApplyTheme を使用します
    C1Theme.ApplyTheme(LayoutRoot, theme)
```

C#

```
// ApplyTheme を使用します
      C1Theme.ApplyTheme(LayoutRoot, theme);
    }
```

アプリケーション全体にテーマを適用する場合は、**System.Windows.ResourceDictionary.MergedDictionaries** プロパティを使用します。まずプロジェクトにテーマアセンブリへの参照を追加して、次のコードのようにテーマを設定します。

Visual Basic

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

    ' Merged Dictionaries を使用します
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
```

```
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();

      // Merged Dictionaries を使用します
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
   }
```

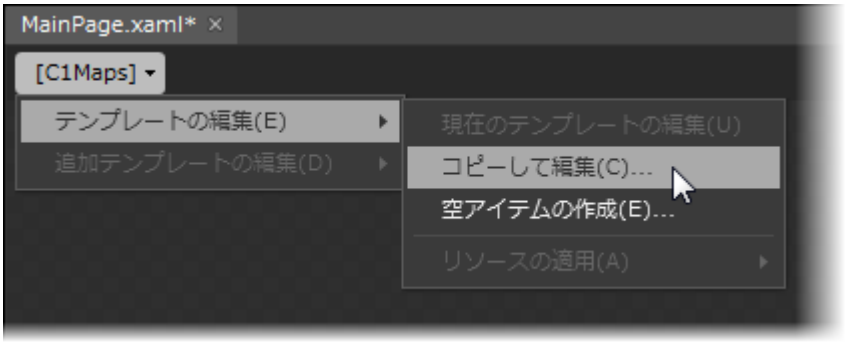
このメソッドは、初めてテーマを適用する場合のみに動作することに注意してください。別の ComponentOne テーマに切り替える場合は、まず **Application.Current.Resources.MergedDictionaries** から以前のテーマを削除してください。

テンプレート

WPF/Silverlight コントロールを使用する主な利点の1つは、これが自由にカスタマイズできるユーザーインターフェイスを持つ「外観のない」コントロールだからです。WPF/Silverlight アプリケーションで独自のユーザーインターフェイス（UI）、つまり「外観」を設計するのと同様に、Maps for WPF/Silverlight によって管理されるデータにも独自の UI を提供できます。Extensible Application Markup Language（XAML。「ザムル」と発音する）は、コードを記述することなく独自の UI を設計するための簡単な方法を提供する XML ベースの宣言型言語です。

テンプレートへのアクセス

テンプレートにアクセスするには、Microsoft Expression Blend で、**C1Maps** コントロールを選択し、メニューから［**テンプレートの編集**］を選択します。［**コピーして編集**］を選択して現在のテンプレートのコピーを作成して編集するか、［**空アイテムの作成**］を選択して新しい空のテンプレートを作成します。



メモ: メニューを使って新しいテンプレートを作成する場合は、テンプレートがそのテンプレートのプロパティに自動的にリンクされます。手作業でテンプレートの XAML を作成する場合は、作成したテンプレートに適切なテンプレートプロパティをリンクする必要があります。

Template プロパティを使ってテンプレートをカスタマイズできます。

外観プロパティ

Maps for WPF/Silverlight には、コントロールの外観をカスタマイズするためのプロパティが含まれます。コントロールに表示されるテキストの外観を変更したり、コントロールのグラフィック要素をカスタマイズすることができます。以下のトピックでは、これらの外観プロパティの一部について説明します。

テキストのプロパティ

次のプロパティを使用して、**C1Maps**コントロールのテキストの外観をカスタマイズできます。

プロパティ	説明
FontFamily	コントロールのフォントファミリを取得または設定します。これは依存プロパティです。
FontSize	フォントサイズを取得または設定します。これは依存プロパティです。
FontStretch	フォントを画面上で伸縮する比率を取得または設定します。これは依存プロパティです。
FontStyle	フォントスタイルを取得または設定します。これは依存プロパティです。
FontWeight	指定されたフォントの太さを取得または設定します。これは依存プロパティです。

色のプロパティ

次のプロパティを使用して、コントロール自体に使用される色をカスタマイズできます。

プロパティ	説明
Background	コントロールの背景を描画するブラシを取得または設定します。これは依存プロパティです。
Foreground	前景色を描画するブラシを取得または設定します。これは依存プロパティです。

境界線のプロパティ

次のプロパティを使用して、コントロールの境界線をカスタマイズできます。

プロパティ	説明
BorderBrush	コントロールの境界線の背景を描画するブラシを取得または設定します。これは依存プロパティです。
BorderThickness	コントロールの境界線の太さを取得または設定します。これは依存プロパティです。

サイズのプロパティ

次のプロパティを使用して、**C1Maps** コントロールのサイズをカスタマイズできます。

プロパティ	説明
Height	要素の高さを取得または設定します。これは依存プロパティです。
MaxHeight	要素の最大の高さを取得または設定します。これは依存プロパティです。
MaxWidth	要素の最大の幅を取得または設定します。これは依存プロパティです。
MinHeight	要素の最小の高さを取得または設定します。これは依存プロパティです。
MinWidth	要素の最小の幅を取得または設定します。これは依存プロパティです。
Width	要素の幅を取得または設定します。これは依存プロパティです。

タスク別ヘルプ

タスク別ヘルプは、ユーザーの皆様が Visual Studio のプログラミングに精通しており、**C1Maps** コントロールを使用する方法を理解していることを前提としています。**Maps for WPF/Silverlight** 製品を初めて使用される場合は、まず「**Maps for WPF/Silverlight** クイックスタート」を参照してください。

このセクションの各トピックは、Maps for WPF/Silverlight 製品を使って特定のタスクを行うための方法を提供します。

また、タスクベースの各ヘルプトピックは、新しい WPF/Silverlight プロジェクトが既に作成されていることを前提としています。

ラベルを追加する

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

```
<c1maps:C1VectorLayer>
<c1maps:C1VectorPlacemark LabelPosition="Left" GeoPoint="-80.107008,42.16389"
StrokeThickness="2" Foreground="#FFEB1212" PinPoint="-80.010866,42.156831"
Label="Erie, PA"/>
</c1maps: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.WPF.C1Maps
```

C#

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

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

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#

```
// 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。

折れ線を追加する

地理座標を折れ線で結ぶには、**C1VectorPolyline** 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 とコードを使用して、3つの点から成る折れ線を作成します。

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

```
<c1maps:C1VectorLayer Margin="2,0,-2,0">
     <c1maps:C1VectorPolyline Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
     </c1maps:C1VectorPolyline>
</c1maps: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. 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.WPF.C1Maps
```

C#

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

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

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
' 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
Dim C1VectorPolyline1 As New C1VectorPolyline()
C1VectorLayer1.Children.Add(C1VectorPolyline1)
' ポイント
C1VectorPolyline1.Points = pcoll
' 外観
C1VectorPolyline1.Stroke = New SolidColorBrush(Colors.Red)
C1VectorPolyline1.StrokeThickness = 3
```

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);
// 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();
v1.Children.Add(C1VectorPolyline1);
  // ポイント
        C1VectorPolyline1.Points = pcoll;
  // 外観
  C1VectorPolyline1.Stroke = new SolidColorBrush(Colors.Red);
  C1VectorPolyline1.StrokeThickness = 3;
```

3. [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、折れ線で結ばれた3つの地理座標を含む **C1Maps** コントロールを示しています。

多角形を追加する

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 とコードを使用して、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

```
<c1maps:C1VectorLayer Margin="2,0,-2,0">
     <c1maps:C1VectorPolygon Points="-80.15,42.12 -123.08,39.09, -3.90,30.85"
StrokeThickness="3" Stroke="Red">
```

```
    </c1maps:C1VectorPolygon>
</c1maps: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.WPF.C1Maps
```

C#

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

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

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#

```
// 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();
v1.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、折れ線で結ばれた3つの地理座標を含む **C1Maps** コントロールを示しています。

マウスオーバー時に地理座標を表示する

このトピックでは、現在のマウス位置の地理座標を返すコードをプロジェクトに追加します。この地理座標は、TextBox コントロールの Text プロパティに文字列として書き込まれます。このタスク別ヘルプトピックは、Visual Studio 2008 で作業していることを前提とします。
次の手順に従います。

1. ツールボックスで、**[StackPanel]**アイコンをダブルクリックして、StackPanel コントロールをプロジェクトに追加します。
2. **StackPanel** コントロールを選択してツールボックスに戻り、**C1Maps** アイコンをダブルクリックしてコントロールを **StackPanel** に追加します。
3. **StackPanel** コントロールを選択してツールボックスに戻り、**TextBox** アイコンをダブルクリックして **TextBox** コントロールを **StackPanel** コントロールに追加します。
4. **StackPanel** のプロパティを次のように設定します。
    a. **Width** プロパティを "**Auto**" に設定します。
    b. **Height** プロパティを "**Auto**" に設定します。
5. **TextBox** コントロールの **Name** プロパティを "ShowCoordinates" に設定します。
6. **C1Maps** コントロールのプロパティを次のように設定します。

a. **Width** プロパティを "350" に設定します。

b. **Height** プロパティを "250" に設定します。

7. **C1Maps** コントロールを選択し、**[プロパティ]**ウィンドウの**[イベント]**ボタン()をクリックします。
8. **[MouseMove]**テキストボックスに「MouseMoveCoordinates」と入力し、[Enter]キーを押して **MouseMoveCoordinates** イベントハンドラをプロジェクトに追加します。
9. コードコメントを次のコードに置き換えます。

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);
```

10. 次の名前空間をインポートします。

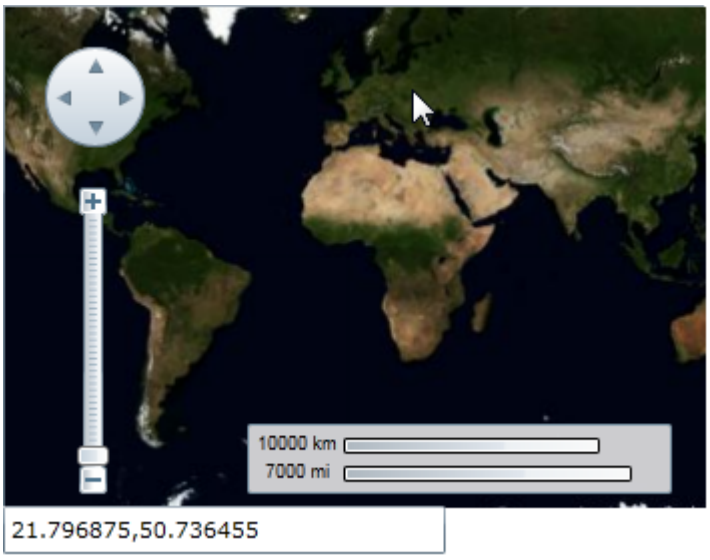
Visual Basic

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

C#

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

11. [F5]キーを押してプロジェクトを実行します。プロジェクトがロードされたら、カーソルをマップ上に移動して、テキストボックスに地理座標が表示されることを確認します。

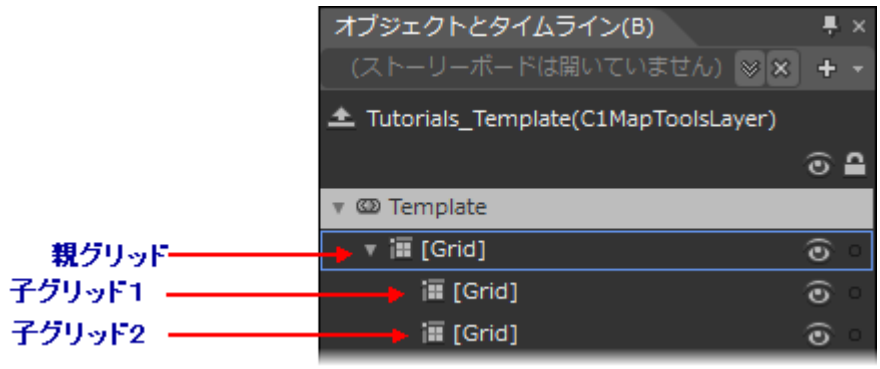


マップツールを並べ替える

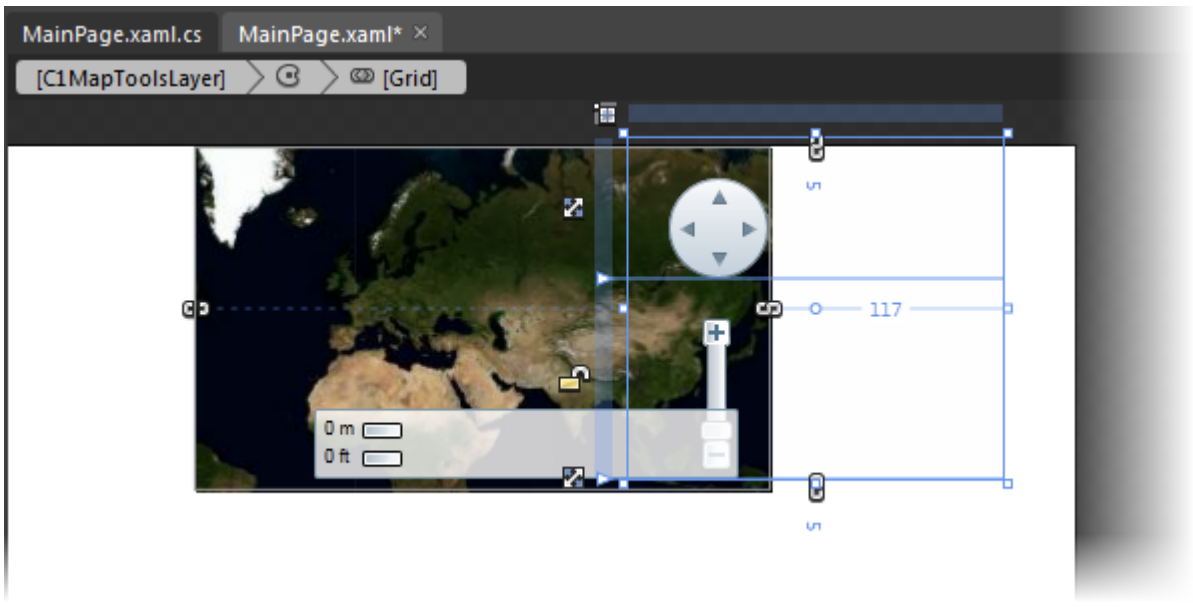
マップツールを変更するには、**C1MapToolsLayer**（詳細については「ツールのカスタマイズ」を参照）とテンプレートを使用します。このトピックは、新しい Microsoft Expression Blend プロジェクトが既に作成されていることを前提としています。Microsoft Expression Blend で C1Maps コントロールの使用方法については、「手順 1：アプリケーションの作成」を参照してください。

次の手順に従います。

1. **C1Maps** を選択して、**[プロパティ]**パネルにプロパティのリストを表示します。
2. **[ツールの表示]**チェックボックスをオフにします。デフォルトのツールが非表示になります。
3. **[レイヤ（コレクション）]**の省略符ボタンをクリックします。**[IMapLayer コレクションエディタ：レイヤ]**ダイアログボックスが開きます。
4. **[別のアイテムを追加]**をクリックして、**[オブジェクトの選択]**ダイアログボックスを開きます。
5. **[C1MapToolsLayer]**を選択し、**[OK]**を押して、**[オブジェクトの選択]**ダイアログボックスを閉じます。
6. **[オブジェクトとタイムライン]**パネルで**[C1MapToolsLayer]**を右クリックし、**[テンプレートの編集]**→**[コピーして編集]**を選択します。テンプレートに「Tutorial Template」という名前を付け、**[OK]**をクリックします。新しいテンプレートが作成されます。**[オブジェクトとタイムライン]**タブに、2つの子グリッドを持つ親グリッドがあることを確認します。

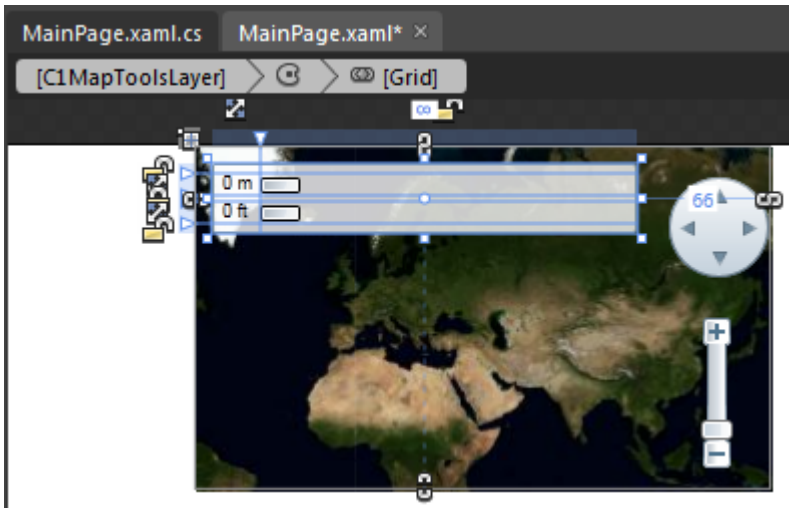


7. **子グリッド1**をクリックして、デザインビューでそのグリッドにフォーカスを置きます。
8. デザインビューで、カーソルを使用して、選択したグリッドをマップの右側に移動します。プロジェクトは次のようになります。



9. **[オブジェクトとタイムライン]**パネルで**子グリッド2**をクリックして、デザインビューでそのグリッドにフォーカスを移動します。
10. デザインビューで、カーソルを使用して、選択したグリッドをコントロールの左上に移動します。デザインビューには次の

ように表示されます。



11. [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。

マップソースを変更する

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 **BingMaps** 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。デフォルトでは、このプロパティは **Bing Maps**(このサービスを使用する前に「法的要件」を参照してください)の航空写真を表示するように設定されていますが、道路ソースまたは混合ソースを表示するように変更することができます。

道路ソースへの変更

次の手順に従います。

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。

Visual Basic

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

C#

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

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

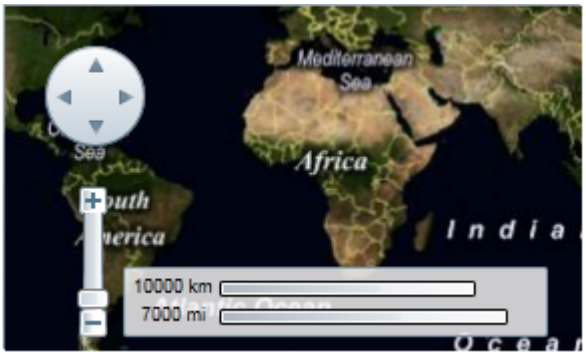
Visual Basic

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

C#

```
c1Maps1.Source = new VirtualEarthRoadSource();
```

3. [F5]キーを押してプログラムを実行し、マップに道路ソースが表示されることを確認します。



混合ソースへの変更

次の手順に従います。

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。

Visual Basic

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

C#

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

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

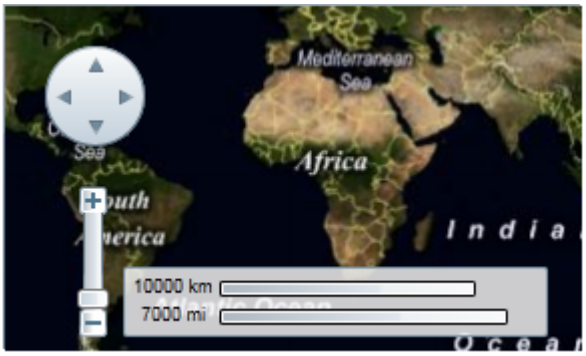
Visual Basic

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

C#

```
c1Maps1.Source = new VirtualEarthHybridSource();
```

3. [F5]キーを押してプログラムを実行し、マップに道路ソースが表示されることを確認します。



マップソースの詳細については、「マップの概念と主要なプロパティ」を参照してください。