ComponentOne

Maps for WPF

Copyright © 2012 ComponentOne LLC. All rights reserved.

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: **info@ComponentOne.com**

Web site: **http://www.componentone.com**

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Maps for WPF Overview....1
- Installing Maps for WPF....1
  - Maps for WPF Setup Files....1
  - Using Maps Powered by Esri....2
  - System Requirements....3
  - Installing Demonstration Versions....4
  - Uninstalling Maps for WPF....4
- End-User License Agreement....4
- Licensing FAQs....4
  - What is Licensing?....4
  - How does Licensing Work?....5
  - Common Scenarios....5
  - Troubleshooting....7
- Technical Support....9
- Redistributable Files....10
- About This Documentation....10

Maps for WPF Key Features....11

Maps for WPF Quick Start....11
- Step 1 of 3: Creating an Application with a C1Maps Control....11
- Step 2 of 3: Binding to a Data Source....12
- Step 3 of 3: Running the Project....15

C1Maps Control Basics....17
- Legal Requirements....17
- HTTPS Support....18
- C1Maps Concepts and Main Properties....18
- Items Layering....19
- Virtualization....21
- Vector Layer....22
  - Vector Objects....22
  - Element Visibility....22

KML Import/Export 23
Data Binding 23
Tool Customization 24

Maps for WPF Layout and Appearance 25
ComponentOne ClearStyle Technology 25
How ClearStyle Works 26
C1Maps ClearStyle Properties 26
C1Maps Themes 27
Maps for WPF Appearance Properties 29
Text Properties 29
Color Properties 30
Border Properties 30
Size Properties 30
Templates 31

Maps for WPF Task-Based Help 33
Adding a Label 33
Adding a Polyline 35
Adding a Polygon 37
Displaying Geographic Coordinates on Mouseover 40
Rearranging the Map Tools 42
Changing the Map Source 44

ComponentOne Maps for WPF Overview

ComponentOne Maps™ for WPF raises the bar on image viewing with smooth zooming, panning, and mapping between screen and geographical coordinates. **C1Maps** allows you to display rich geographical information from various sources, including Bing Maps™ and Google Maps™.

Built on top of the Microsoft Deep Zoom technology, **C1Maps** enables end-users to enjoy extreme close-ups with high-resolution images and smooth transitions. It also supports layers that allow you to superimpose your own custom elements to the maps.

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

- C1Maps Control Basics (page 17)

- Quick Start (page 11)

- Task-Based Help (page 33)

Installing Maps for WPF

The following sections provide helpful information on installing **ComponentOne Maps for WPF**.

Maps for WPF Setup Files

The installation program will create the directory **C:\Program Files\ComponentOne\Studio for WPF**, which contains the following subdirectories:

Bin — Contains copies of all ComponentOne binaries (DLLs, EXEs). For **Component Maps for WPF**, the following DLLs are installed:

- C1.WPF.dll
- C1.WPF.Maps.dll
- C1.WPF.Maps.Expression.Design.dll
- C1.WPF.Maps.Expression.Design.4.0.dll
- C1.WPF.Maps.VisualStudio.Design.dll
- C1.WPF.Maps.VisualStudio.Design.4.0.dll

In addition, the following files from the Microsoft WPF Toolkit are also installed:

- WPFToolkit.dll
- WPFToolkit.Design.dll
- WPFToolkit.VisualStudio.Design.dll

For more information about the Microsoft WPF Toolkit, see CodePlex. The C1.WPF.dll and WPFToolkit.dll assemblies are required for deployment.

The **ComponentOne Studio for WPF Help Setup** program installs integrated Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for WPF\HelpViewer folder:

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

Windows XP path: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\<username>\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

Common	Contains support and data files that are used by many of the demo programs.
Studio for WPF	Contains samples for **DateTimeEditors for WPF**.

You can access samples from the ComponentOne Control Explorer**. To view samples, on your desktop, click the** Start **button and then click** ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.**

Esri Maps

Esri® files are installed with **ComponentOne Studio for Silverlight**, **ComponentOne Studio for WPF**, and **ComponentOne Studio for Windows Phone** by default to the following folders:

32-bit machine : C:\Program Files\ESRI SDKs\<Platform>\<version number>

64-bit machine: C:\Program Files (x86)\ESRI SDKs\<Platform>\<version number>

Files are provided for multiple languages, including: English, German (de), Spanish (es), French (fr), Italian (it), Japanese (ja), Portuguese (pt-BR), Russian (ru) and Chinese (zh-CN).

See Using Maps Powered by Esri (page 2) or visit the Esri website at http://www.esri.com for additional information.

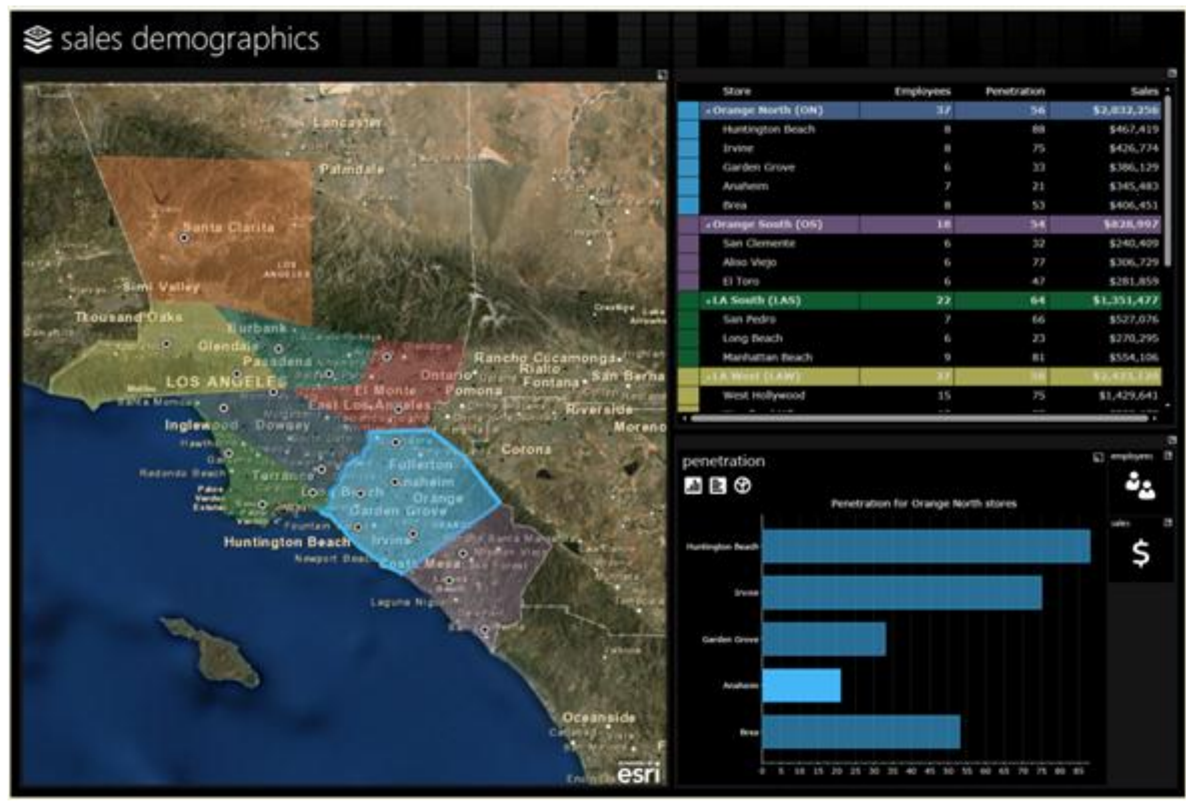
Using Maps Powered by Esri

Easily transform GIS data into business intelligence with controls for Silverlight, WPF, and Windows Phone powered by Esri® software.

By using the ComponentOne award-winning UI controls, you'll have the tools you need to seamlessly create rich, map-enabled user interfaces.

Benefits of Maps powered by Esri:

- Esri knows maps: Esri is the leading online map and GIS provider.
- Maps are technical: Using maps within your application is a very technical thing, so you don't want to take your chance using anyone but the best.
- Company of choice: Esri is the company of choice of many top companies and government agencies.
- Fulfill any developers' mapping needs: Esri mapping tools are flexible and will fill the needs of any mapping solution.



E

sri Map Example

There are no additional charges for using the Esri maps included with ComponentOne products. Simply create a free online account at http://www.arcgisonline.com to start taking advantage of the Esri map controls. Esri licensing terms can be found in our Licensing Information and End User Licensing Agreement at http://www.componentone.com/SuperPages/Licensing/.

To learn more about Esri and Esri maps, please visit Esri at http://www.esri.com. There you will find detailed support, including documentation, forums, samples, and much more.

See the Studio for WPF Setup Files (page 1) topic for more information on the Esri files installed with this product.

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2)
	Windows Vista™
	Windows 7
	Windows 2008 Server
Environments:	.NET Framework 3.5 or later
	Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP
	Visual Studio® 2008 or later

Microsoft® Expression® Blend Compatibility:	**Maps for WPF** includes design-time support for Expression Blend.

Note: The **C1.WPF.VisualStudio.Design.dll** assembly is required by Visual Studio and the **C1.WPF.Expression.Design.dll** assembly is required by Expression Blend. The **C1.WPF.Expression.Design.dll** and **C1.WPF.VisualStudio.Design.dll** assemblies installed with **Maps for WPF** should always be placed in the same folder as **C1.WPF.dll**; the DLLs should NOT be placed in the Global Assembly Cache (GAC).

Installing Demonstration Versions

If you wish to try **ComponentOne Maps for WPF** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so that a ComponentOne banner will not appear when your users run the applications.

Uninstalling Maps for WPF

To uninstall **ComponentOne Maps for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for WPF** and click the **Remove** button.
3. Click **Yes** to remove the program.

To uninstall **ComponentOne Studio for WPF** integrated help:

1. Open the Control Panel and select Add or Remove Programs (Programs and Features in Windows 7/Vista).
2. Select ComponentOne Studio for WPF Help and click the Remove button.
3. Click **Yes** to remove the integrated help.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

> **Note:** The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license.
- A "licenses.licx" file that contains the licensed component strong name and version information.

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the **App_Licenses.dll** assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the **App_licenses.dll** must always be deployed with the application.

The **licenses.licx** file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the **licenses.licx** file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox or, from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the **licenses.licx** file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a **licenses.licx** file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the **licenses.licx** file and things will then work as expected. (The component can be removed from the form after the **licenses.licx** file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the **licenses.licx** file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a **LicenseProvider** attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the **licenses.licx** file and the base class will handle the licensing process as usual. No additional work is needed. For example:

  ```
  [LicenseProvider(typeof(LicenseProvider))]
  class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
  {
    // ...
  }
  ```

- Add an instance of the base component to the form.

  This will embed the licensing information into the **licenses.licx** file as in the previous scenario and the base component will find it and use it. As before, the extra instance can be deleted after the **licenses.licx** file has been created.

Please note that ComponentOne licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design-time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to and therefore Visual Studio won't create a **licenses.licx** file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the **licenses.licx** file into the console application project.

Make sure the **licenses.licx** file is configured as an embedded resource. To do this, right-click the **licenses.licx** file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the **licenses.licx** is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an EXE file and also a licenses.licx file with licensing information in it.
2. Copy the **licenses.licx** file from the application directory to the target folder (**Debug** or **Release**).
3. Copy the **C1Lc.exe** utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use **C1Lc.exe** to compile the **licenses.licx** file. The command line should look like this:
   ```
   c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
   ```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select **Properties**, and go to the **Linker/Command Line** option. Enter the following:
   ```
   /ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
   ```
6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the **AssemblyConfiguration** attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design-time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
   public class MyDerivedControl : C1LicensedControl
   {
         // ...
   }
```

Note that the **AssemblyConfiguration** string may contain additional text before or after the given string, so the **AssemblyConfiguration** attribute can be used for other purposes as well. For example:

```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design-time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the **licenses.licx** file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another **licenses.licx** file or follow the alternate procedure following.
5. Save the file, then close the **licenses.licx** tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a **licenses.licx** file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the **licenses.licx** file and right-click it.
3. Select the **Rebuild Licenses** option (this will rebuild the **App_Licenses.licx** file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 – Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

Option 2 – Continue to use the components you have.

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/SuperProducts/SupportServices/.

Some methods for obtaining technical support include:

- **Online Resources**
  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.

- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Product Forums**
  ComponentOne's product forums are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.

- **Documentation**
  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

> **Note:** You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne Maps for WPF is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.dll
- C1.WPF.Maps.dll
- C1.WPF.Maps.Expression.Design.dll
- C1.WPF.Maps.VisualStudio.Design.dll

In addition, the following file from the Microsoft WPF Toolkit is also installed and is redistributable:

- WPFToolkit.dll

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About This Documentation

Acknowledgements

Microsoft, Windows, Windows Vista, and Visual Studio, and Silverlight, are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Firefox is a registered trademark of the Mozilla Foundation. Safari is a trademark of Apple Inc., registered in the U.S. and other countries.

Esri is a registered trademark of Environmental Systems Research Institute, Inc. (Esri) in the United States, the European Community, or certain other jurisdictions.

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA
412.681.4343
412.681.4384 (Fax)

http://www.componentone.com

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

Maps for WPF Key Features

ComponentOne Maps for WPF allows you to create customized, rich applications. Make the most of **Maps for WPF** by taking advantage of the following key features:

- **Draw any Geometry**

  C1Maps" vector layer allows you to draw geometries/shapes/polygons/paths with geo coordinates on top of the map. The vector layer is useful to draw:

  - Political borders (such as countries or states)
  - Geo details (for example, showing automobiles or airplane routes)
  - Choropleth maps (based on statistical data, such as showing population per country)

  You can use the vector layer instead of the regular Microsoft Virtual Earth source to show a world map representation.

- **KML Support**

  The vector layer supports basic KML import/export (KML is the standard file format to exchange drawings on top of maps). For more information, see KML Import/Export (page 23).

- **Rich Geographical Information**

  Display rich geographical information from various sources, including Bing Maps as well as any custom source. For example, you can build your own source for Yahoo! Maps.

- **Display a Large Number of Elements on the Map**

  Maps for WPF allows virtualization of local and server data. Using its virtual layer Maps only displays and requests the elements currently visible.

- **Zoom, Pan, and Map Coordinates**

  Maps for WPF supports zooming and panning using the mouse or the keyboard. It also supports mapping between screen and geographical coordinates.

- **Layers Support**

  Use layers to add your own custom elements to the maps. Elements are linked to geographical locations. For more information, see Vector Layer (page 22), Virtualization (page 21), and Items Layering (page 19).

Maps for WPF Quick Start

The following quick start guide is intended to get you up and running with **Maps for WPF**. You'll start in Expression Blend to create a new project with the C1Maps control. Once the control has been added, you will customize its appearance, add a C1VectorLayer and a C1VectorPlacemark to it, create a data source, and then bind properties of the C1VectorPlacemark to the data source. At the end of this quick start, you'll have a fully functional map control that contains a series of labeled placemarks.

Step 1 of 3: Creating an Application with a C1Maps Control

In this step, you'll begin in Expression Blend to create a WPF application using the C1Maps control. You will also set the control's properties.

Complete the following steps:

1. In Expression Blend, select **File | New Project**.

2. In the **New Project** dialog box, select the WPF project type in the left pane and, in the right-pane, select **WPF Application + Website**.
3. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**. Blend creates a new application, which opens with the **MainPage.xaml** file displayed in Design view.
4. Add a reference to the **C1.WPF.Maps** assembly by completing the following steps:
    a. Select **Project | Add Reference**.
    b. Browse to find **the C1.WPF.Maps.dll** assembly installed with **Maps for WPF**.

    > **Note**: The **C1.WPF.Maps.dll** file is installed to C:\Program Files\ComponentOne\Studio for WPF\bin by default.

    c. Select **C1.WPF.dll** and click **Open**. A reference is added to your project.
5. Add the C1Maps control to your project by completing the following steps:
    a. On the menu, select **Window | Assets** to open the **Assets** tab.
    b. Under the **Assets** tab, enter "C1Maps" into the search bar.
    c. The C1Maps control's icon appears.
    d. Double-click the C1Maps icon to add the control to your project.
6. In the **Objects and Timeline** panel, select **[C1Maps]** and then, under the **Properties** panel, set the following properties:
    - Set the **Name** property to "C1Maps1" so that your control will have a unique identifier to call in code.
    - Set the **Width** property to "405".
    - Set the **Height** property to "472".
    - Set the Zoom property to "2" to set the zoom factor to 2x the original zoom.
    - Set the Center property to "-65, -25" so that only South America appears on the map.

In this step, you created a Blend WPF project and added a C1Maps control to it; in addition, you set the properties of the C1Maps control.

Step 2 of 3: Binding to a Data Source

In this step, you will create a class with two properties, **Name** and **LatLong**, and populate them with an array collection. In addition, you will add a C1VectorLayer containing a C1VectorPlacemark to the control. You will then bind the **Name** property to the C1VectorPlacemark"s Label property and the **LatLong** property to the C1VectorPlacemark"s GeoPoint property.

Complete the following steps:

1. Open the **MainPage.xaml** code page (this will be either **MainPage.xaml.cs or MainPage.xaml.vb** depending on which language you've chosen for your project).
2. Add the following class to your project, placing it beneath the namespace declaration:
3. This class creates a class with two properties: a string property named **Name** and a **Point** property named **LongLat**.
    - Visual Basic

    ```
    Public Class City
    Private _LongLat As Point
    ```

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

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

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

- C#

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

4. Add the following code beneath the **InitializeComponent()** method to create the array collection that will populate the **Name** property and the **LongLat** property:

- 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")
        }
C1Maps1.DataContext = cities
```

- C#

```
City[] cities = new City[]
{
   new City(){ LongLat= new Point(-58.40, -34.36), Name="Buenos
Aires"},
   new City(){ LongLat= new Point(-47.92, -15.78), Name="Brasilia"},
   new City(){ LongLat= new Point(-70.39, -33.26), Name="Santiago"},
   new City(){ LongLat= new Point(-78.35, -0.15), Name="Quito"},
   new City(){ LongLat= new Point(-66.55, 10.30), Name="Caracas"},
   new City(){ LongLat= new Point(-56.11, -34.53), Name="Montevideo"},
   new City(){ LongLat= new Point(-77.03, -12.03), Name="Lima"},
   new City(){ LongLat= new Point(-57.40, -25.16), Name="Asuncion"},
   new City(){ LongLat= new Point(-74.05, 4.36), Name="Bogota"},
   new City(){ LongLat= new Point(-68.09, -16.30), Name="La Paz"},
   new City(){ LongLat= new Point(-58.10, 6.48), Name="Georgetown"},
   new City(){ LongLat= new Point(-55.10, 5.50), Name="Paramaribo"},
};
C1Maps1.DataContext = cities;
```

5. Switch to XAML view and change the `<c1:C1Maps>` markup so that it has a beginning and a closing tag so that it looks as follows:

```
<c1:C1Maps x:Name="C1Maps1" FadeInTiles="False" Margin="0,0,235,8"
TargetCenter="-65,-25" Center="-58,-25" Zoom="2">
</c1maps>
```

6. Add `Foreground="Aqua"` to the `<c1:C1Maps>` tag.
7. Place the following XAML markup between the `<c1:C1Maps>` and `</c1:C1Maps>` tags:

```
<c1:C1Maps.Resources>
<!—Item template →
   <DataTemplate x:Key="templPts">
     <c1:C1VectorPlacemark
       GeoPoint="{Binding Path=LongLat}" Fill="Aqua" Stroke="Aqua"
       Label="{Binding Path=Name}" LabelPosition="Top" >
       <c1:C1VectorPlacemark.Geometry>
         <EllipseGeometry RadiusX="2" RadiusY="2" />
       </c1:C1VectorPlacemark.Geometry>
     </c1:C1VectorPlacemark>
   </DataTemplate>
</c1:C1Maps.Resources>
<c1:C1VectorLayer ItemsSource="{Binding}"
    ItemTemplate="{StaticResource templPts}" HorizontalAlignment="Right"
Width="403" />
```

This XAML creates a data template, a C1VectorPlacemark, and a C1VectorLayer. The C1VectorLayer's ItemsSource property is bound to the entire data source, and the C1VectorPlacemark''s GeoPoint property is bound to the value of the **LongLat** property while its Label property is set to the value of the **Name** property. When you run the project, the Label and **Name** properties will be populated by the data source to create a series of labeled placemarks on the map.

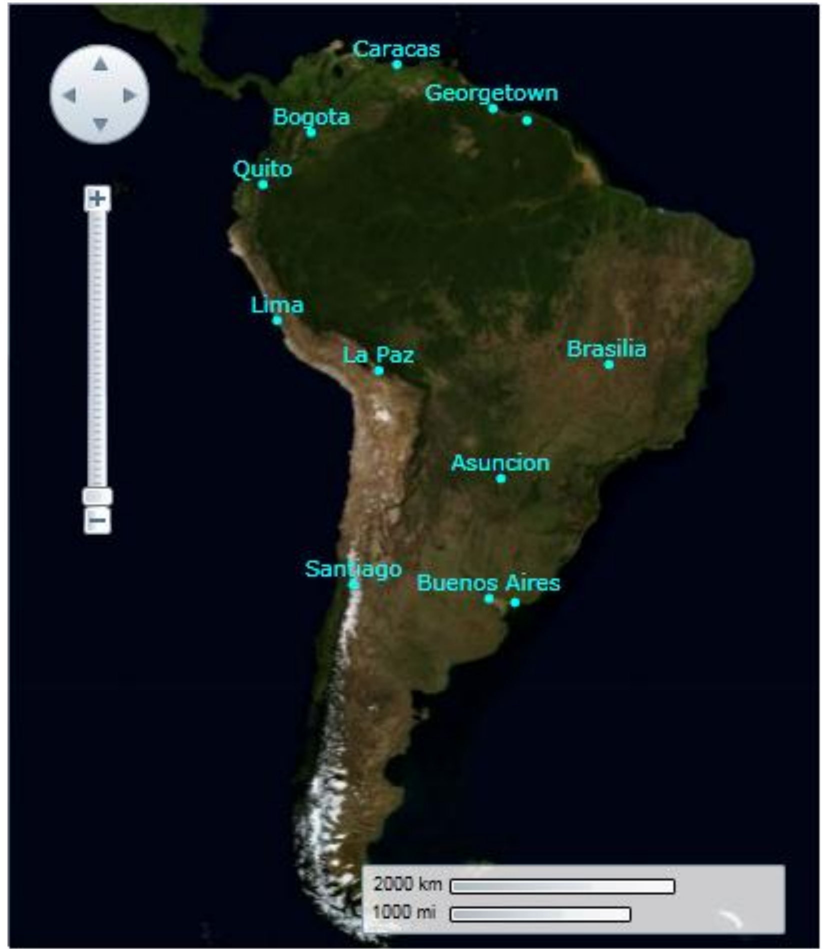
In this step, you created a data source and bound it to the properties of the C1VectorPlacemark. In the next step, you'll run the program and view the results of the quick start project.

Step 3 of 3: Running the Project

In the previous steps, you created a WPF project with a C1Maps control, created a data source, added a C1VectorLayer and a C1VectorPlacemark to the C1Maps control, and then bound the data source to properties of the C1VectorPlacemark.

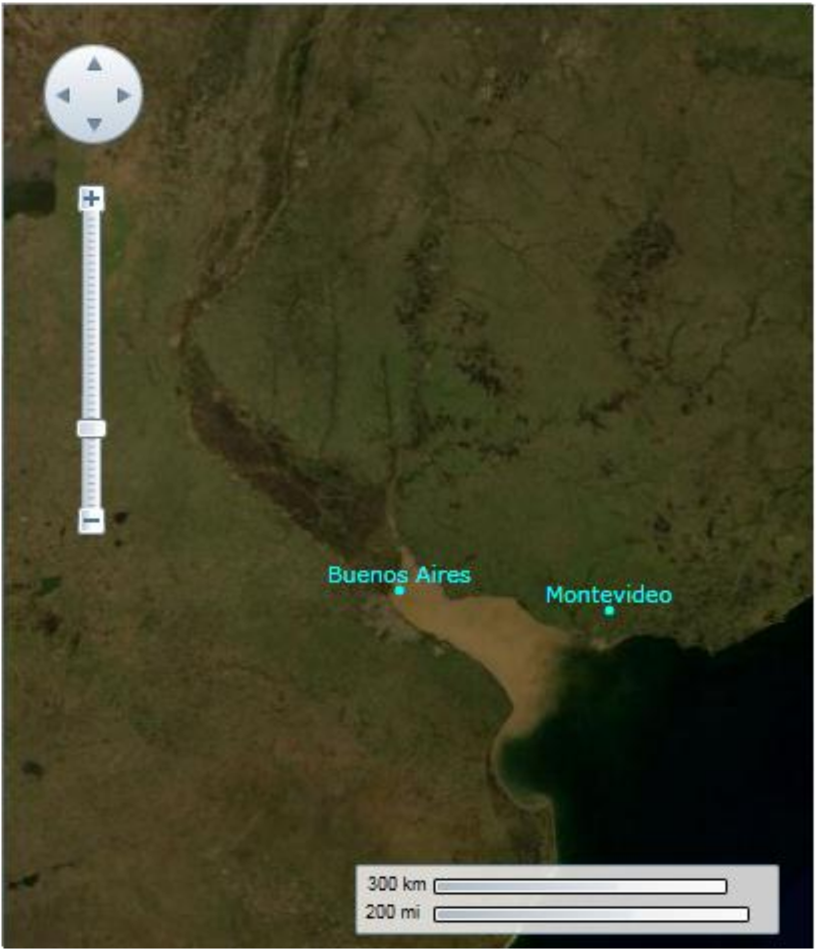
Complete the following steps:

1. Press F5 to run the project and observe that the C1Maps control appears as follows:



Observe that there are two dots, one in the vicinity of **Buenos Aires** and the other in the vicinity of **Georgetown**, that don't have names next to them.

2. Double-click in the area of **Buenos Aires**. Repeat this step twice and observe that another label, one marking **Montevideo**, appears on the map.



Congratulations! You have completed the **Maps for WPF** quick start. We recommend that you continue to familiarize yourself with the control by visiting the C1Maps Control Basics (page 17) and Maps for WPF Task-Based Help (page 33) sections of the Help file.

C1Maps Control Basics

The **C1.WPF.Maps** assembly contains the **C1Maps** control, which displays rich geographical information from various sources, including Bing Maps, as well as your own custom data.

C1Maps supports zooming, panning, and mapping between screen and geographical coordinates. It also supports layers that allow you to superimpose elements on the maps. The layers support item virtualization and allow you to display static elements as well as elements that are attached to geographical locations.

The following topics introduce you to the basics of the C1Maps control.

Legal Requirements

C1Maps allows you to use geographical information from **Bing Maps**™. Before using this service, you should check the licensing requirements associated with it. These licensing terms can be found at:

- http://www.microsoft.com/virtualearth/product/licensing.aspx

HTTPS Support

Microsoft Silverlight restricts cross-zone, cross-domain, and cross-scheme URL access for security reasons. The following table summarizes these rules:

	Downloader object	Media, images, ASX	XAML files, Font files	Streaming media
Allowed schemes	HTTP, HTTPS	HTTP, HTTPS, FILE	HTTP, HTTPS, FILE	HTTP
Cross-scheme access	No	No	No	Not from HTTPS
Cross-Web domain access	No	If not HTTPS	No	Yes
Cross-zone access (Windows)	No	No	No	No
Cross-zone access (Macintosh)	No	Yes	No	Yes
Redirection allowed	Same domain (Firefox/Safari only)	Same domain	Same domain	No

For more detailed information on Silverlight HTTPS support, visit the **Silverlight URL Access Policy** on MSDN, which is located at http://msdn.microsoft.com/en-us/library/bb820909.aspx.

> **Note**: It is possible to use a C1Maps control with HTTPS; however, the image tiles *must* come from the same domain as the Silverlight application.

C1Maps Concepts and Main Properties

This section details basic **C1Maps** concepts and describes the main properties.

Map Source

C1Maps can display geographical information from several sources. By default, **C1Maps** uses **BingMaps** aerial photographs as the source, but you can change that using the **Source** property, which takes an object of type **MultiScaleTileSource**.

The following sources are included:

- Virtual Earth Aerial Source
  - Visual Basic
    ```
    map1.Source = new VirtualEarthAerialSource()
    ```
  - C#
    ```
    map1.Source = new VirtualEarthAerialSource();
    ```
- Virtual Earth Road Source
  - Visual Basic
    ```
    map2.Source = new VirtualEarthRoadSource()
    ```
  - C#
    ```
    map2.Source = new VirtualEarthRoadSource();
    ```
- Virtual Earth Hybrid Source
  - Visual Basic
    ```
    map3.Source = new VirtualEarthHybridSource()
    ```

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

Visible Map

The portion of the map that is currently visible is determined by the **Center** and **Zoom** properties, and by the size of the control:

The **Center** property is of type **Point** but it actually represents a geographic coordinate in which the X property is longitude and the Y property is latitude. The user can change the value of the **Center** property by dragging the map with the mouse, or by using the navigator control shown on the left top corner.

The **Zoom** property indicates the current resolution of the map. A zoom value of 0 has the map totally zoomed out, and each increment of 1 doubles the map resolution. The user can change the value of the Zoom property using the mouse wheel or the zoom control on the left side of the control.

Coordinate Systems

C1Maps uses three coordinate systems:

- **Geographic** coordinates mark points in the world using latitude and longitude. This coordinate system is not Cartesian, which means the scale of the map may change as you pan.
- **Logical** coordinates go from 0 to 1 on each axis for the whole extent of the map, and they are easier to work with because they are Cartesian coordinates.
- **Screen** coordinates are the pixel coordinates of the Control relative to the top-left corner. These are useful for positioning items within the control and for handling mouse events.

C1Maps provides four methods for converting between these coordinate systems: **ScreenToGeographic**, **ScreenToLogic**, **GeographicToScreen**, and **LogicToScreen**. The conversion between geographic and logic coordinates is done by the projection configured using the **C1Maps.Projection** property. The projection can be changed to support a different map, the default is the Mercator projection used by **LiveMaps** and most other providers.

Information Layers

In addition to the geographical information provided by the source, you can add layers of information to the map. **C1Maps** includes five layers by default:

- **C1MapItemsLayer** is the layer used to display arbitrary items positioned geographically on the map. This layer is an **ItemsControl**, so it supports directly adding **UIElement** objects or generic data objects with a **DataTemplate** that can convert them into visual items.
- **C1MapVirtualLayer** displays items that are virtualized; this means they are only loaded when the region of the map they belong to is visible. It also supports asynchronous requests, so that new items can be downloaded from the server only when they come into view.
- **C1VectorLayer** displays vector data, like lines and polygons, whose vertices are geographically positioned. It can save and load data from KML files.
- **C1MapToolsLayer** is the next layer, used to display tools for panning and zooming, and a scale. This layer is built into **C1Maps'** template, so it's not necessary to add it manually.
- **C1MapTilesLayer** is the background layer where the map tiles are displayed. You normally don't have to use this layer because it is managed by **C1Maps** automatically.

Items Layering

C1MapItemsLayer is the easiest way to display items over a map. It inherits from **ItemsControl** so it supports directly adding **UIElement** objects or generic data objects with a **DataTemplate** that can convert them into visual items. Elements added to a **C1MapItemsLayer** are positioned using the **C1MapCanvas.LatLong** attached property. Let's look at a sample:

```
<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>
```

This creates a **C1Maps** control in XAML and adds a **C1MapItemsLayer** to its **Layers** collection. Any number of layers can be added to the **Layers** collection, they will be displayed one on top of the other.

We add one item to the items layer, an ellipse positioned at latitude/longitude (40.4587, -79.9247). Note that these numbers are in reverse order in XAML. This is because **LatLong** values are represented by a **Point** structure with its X value corresponding to longitude and its Y value corresponding to latitude (this matches the way maps and X/Y axis are usually oriented).

In the previous example we can also see the **C1MapCanvas.Pinpoint** attached property in use. This property configures which point inside the element will match the geographic coordinates set in the **LatLong** property. In the example case, **Pinpoint** is set to (10, 10) so that the ellipse will be centered on the LatLong position.

Let's look at a second example. This time we will create a **C1Maps** control in code, and populate it with data. We will use the following class:

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

And here is the example code:

```
var map = new C1Maps();
var itemsLayer = new C1MapItemsLayer
{
    ItemsSource = new[]
    {
        new Place {
            Name = "ComponentOne",
            LatLong = new Point(-79.92476,  40.45873), },
        new Place {
            Name = "Greenwich Park",
            LatLong = new Point(  0.00057,  51.47617), },
    },
    ItemTemplate = itemTemplate
};
map.Layers.Add(itemsLayer);
```

We populate the **ItemsSource** with instances of the **Place** class, and we set **ItemTemplate** to the following **DataTemplate** defined in the Page's resources:

```
<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>
```

This **DataTemplate** binds **C1MapCanvas.LatLong** to the **LatLong** defined in the items and displays the place's Name in a **TextBlock**.

Using **ItemTemplate** and **ItemsSource** it's easy to load data from a database. You only have to setup a Web service returning a collection of data objects, set the collection as **ItemsSource**, and create a **DataTemplate** binding the appropriate values.

Virtualization

C1MapVirtualLayer displays elements over the map supporting virtualization and asynchronous data loading. It can be used to display an unlimited number of elements, as long as not many of them are visible at the same time. Its object model is quite different from **C1MapItemsLayer**; **C1MapVirtualLayer** requires a division of the map space in regions, and the items' source must implement the **IMapVirtualSource** interface.

The division of map space is defined using the **C1MapVirtualLayer.Slices** collection of **MapSlice**. Each map slice defines a minimum zoom level for its division, and the maximum zoom level for a slice is the minimum zoom layer of the next slice (or, if it is the last slice, its maximum zoom level is the maximum zoom of the map). In turn, each slice is divided in a grid of latitude/longitude divisions.

Take the following layer as an example:

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

There are two slices: one goes from zoom 5 to 10, and the other one from zoom 10 to the maximum zoom. When the zoom value moves from one slice to another, the virtual layer will request data from its source. Also, the first slice has a 2 by 2 lat/long division; this means that map is divided in 4 regions, and the layer only requests data for the current visible regions. The second slice is divided into 16 regions, higher zoom values require more divisions to perform well.

To understand the **IMapVirtualSource** interface, let's look at an implementation from the Factories sample:

```
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);
    }
}
```

The **Request** method receives a region of the map space as parameter, and expects a collection of items to be returned using a callback. This particular implementation first checks if the minimal zoom requested is less than an application parameter, if true it does nothing. Otherwise, it calls a Web service to obtain the data.

Server-side we have the implementation of **GetStores**. It iterates through all the elements in a database, and returns the items that are inside the bounds requested:

```
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;
}
```

A better implementation should have the stores already divided in regions to prevent iterating through all of them.

Vector Layer

The Vector layer allows you to place various objects with geographic coordinates on the map.

Vector Objects

The following main vector elements that can be used on the vector layer:

- **C1VectorPolyline** – similar to Polygon class, except that this object needn't be a closed shape. The polyline is formed using geographical coordinates. Typical usage: paths, routes. For task-based help, see Adding a Polyline (page 35).
- **C1VectorPolygon** – similar to Polyline class, but it draws a polygon, which is a connected series of lines that form a closed shape. The polygon is formed using geographical coordinates. Typical usage: borders, regions. For task-based help, see Adding a Polygon (page 37).
- **C1VectorPlacemark** – an object attached to the geographical point. The placemarks have scale-independent geometry which coordinates are expressed in pixel coordinates and optional label (any UIElement). Typical usage: labels, icons, marks on the map. For task-based help, see Adding a Label (page 33).

Element Visibility

There are several properties that can control element visibility depending on the current map scale. For example, you can show more details when zooming in and hide them when zooming out.

The global control is performed by **C1VectorLayer.MinSize** property that specifies at which minimal linear screen size the element becomes visible.

There is a special property that controls the visibility of **C1VectorPlacemark** labels. **C1VectorLayer.LabelVisibilty** can have the following values:

- **Hide** – labels are not visible, they are shown as ToolTips.
- **AutoHide** – overlapped labels are hidden.
- **Visible** – all labels are visible.

Additionally, each vector element can have its own visibility settings which are stored in **LOD** property, and it has priority over the global values.

LOD (Level ff Details) structure has the following properties:

- **MinSize**, **MaxSize** – specifies the visible range of linear screen size of an element, if the size does not fit in the range the element is hidden.
- **MinZoom**, **MaxZoom** – alternatively you can specify the range of map scales (**C1Maps.Zoom** property) in which the element should be displayed.

KML Import/Export

KML is an XML-based language for geographic visualization and annotation which was originally created for Google Earth. For more information, see http://code.google.com/apis/kml/documentation.

KML import is performed by the **KmlReader** class which has static methods that create collection of vector objects from the supplied KML source (string or stream). The collection can be easily added to the **C1VectorLayer**. The **DataContext** of the imported object is set to the corresponding **XElement** from the KML source so you can use the original element to perform custom operation during import.

Import limitations:

- Only KML Placemark elements are supported.
- Inner polygons are not supported.
- Icons are not supported.
- External links are not supported.

KML export is performed by **KmlWriter** class which has static methods that write the collection of vector objects to the provided stream in KML format.

The **KmlWriter.Write()** method has the parameter saveElementCallback that allows you to perform custom operations during export. The method is called for each element that is saved in KML stream. For example, using the callback method you can add KML custom data to the elements.

Export limitation:

- **C1VectorPlacemark.Geometry** is not saved in KML stream.

Data Binding

C1VectorLayer has two properties to support data binding:

- **ItemsSource** – specifies a collection of source objects.
- **ItemTemplate** – specifies the appearance of each object on the layer. The Item template must define the class which is inherited from **C1VectorItemBase**.

Data Binding Example

Suppose you have a collection of **City** objects:

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

The template defines how to create **C1VectorPlacemark** from the **City** class.

```
<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>
```

Finally, you need to use some real collection as a data source.

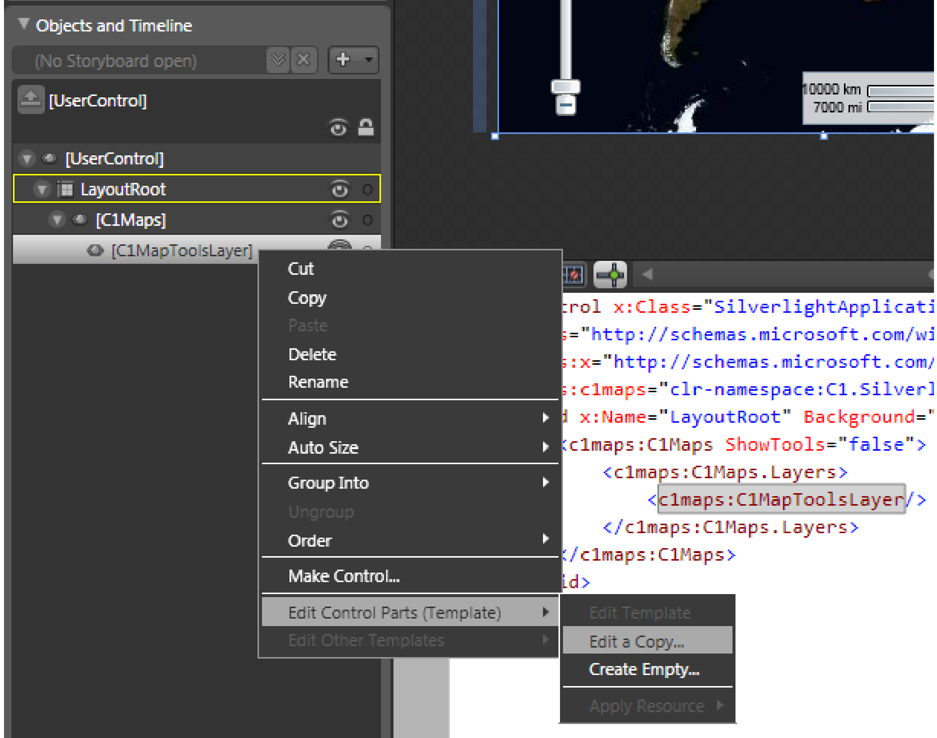
```
City[] cities = new City[]
{
    new City(){ LongLat= new Point(30.32,59.93), Name="Saint Petersburg"},
    new City(){ LongLat= new Point(24.94,60.17), Name="Helsinki"},
    new City(){ LongLat= new Point(18.07,59.33), Name="Stockholm"},
    new City(){ LongLat= new Point(10.75,59.91), Name="Oslo"},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

maps.DataContext = cities;
```

Tool Customization

The panning and zooming tools displayed by default in the map are implemented in the **C1MapToolsLayer**. It is included in **C1Maps**' template, so it's not necessary to add it to the Layers collection. To customize the tools you will first hide the default tools by setting **C1Maps.ShowTools** to **False**, and then add your own **C1MapToolsLayer** instance. Here is the XAML for this:

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

Note that you could also implement a totally different layer for the tools, but you'll just modify the template of the built-in tools in this example. Now, to edit this XAML in Blend, you can right-click the **ToolsLayer** and select **Edit Control Parts (Template) | Edit a Copy**:



Now you can just edit the template in Blend, and the changes will be reflected in the map.

Maps for WPF Layout and Appearance

The following topics detail how to customize the C1Maps control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

ComponentOne ClearStyle Technology

ComponentOne ClearStyle™ technology is a new, quick and easy approach to providing Silverlight and WPF control styling. ClearStyle allows you to create a custom style for a control without having to deal with the hassle of XAML templates and style resources.

Currently, to add a theme to all standard WPF controls, you must create a style resource template. In Microsoft Visual Studio, this process can be difficult; this is why Microsoft introduced Expression Blend to make the task a bit easier. Having to jump between two environments can be a bit challenging to developers who are not familiar with Blend or do not have the time to learn it. You could hire a designer, but that can complicate things when your designer and your developers are sharing XAML files.

That's where ClearStyle comes in. With ClearStyle the styling capabilities are brought to you in Visual Studio in the most intuitive manner possible. In most situations you just want to make simple styling changes to the controls in your application so this process should be simple. For example, if you just want to change the row color of your data grid this should be as simple as setting one property. You shouldn't have to create a full and complicated-looking template just to simply change a few colors.

How ClearStyle Works

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

C1Maps ClearStyle Properties

Maps for WPF supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

The following table outlines thebrush properties of the **C1Maps** control:

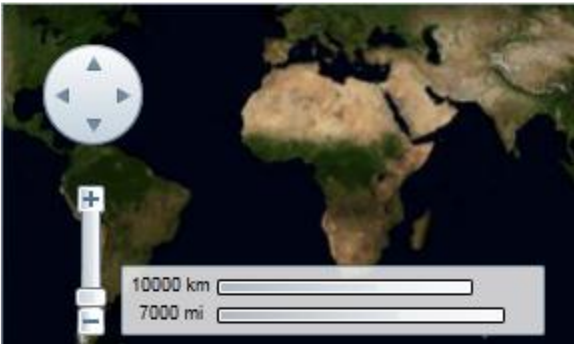
Brush	Description
Background	Gets or sets the brush of the control’s background.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the map buttons when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when they are clicked on.

You can completely change the appearance of the **C1Maps** control by setting a few properties, such as the **Background** property, which sets the background color of the map’s tools. For example, if you set the **Background** property to "##FFE40005", the **C1Maps** control would appear similar to the following:

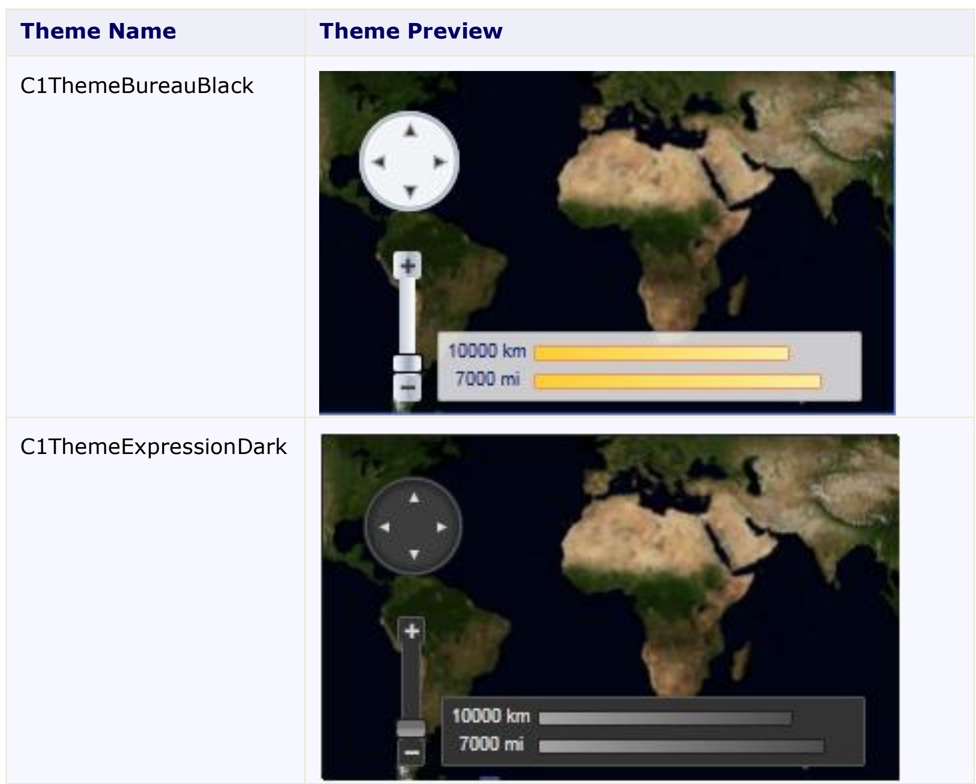
It’s that simple with ComponentOne’s ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology (page 25) topic.

C1Maps Themes

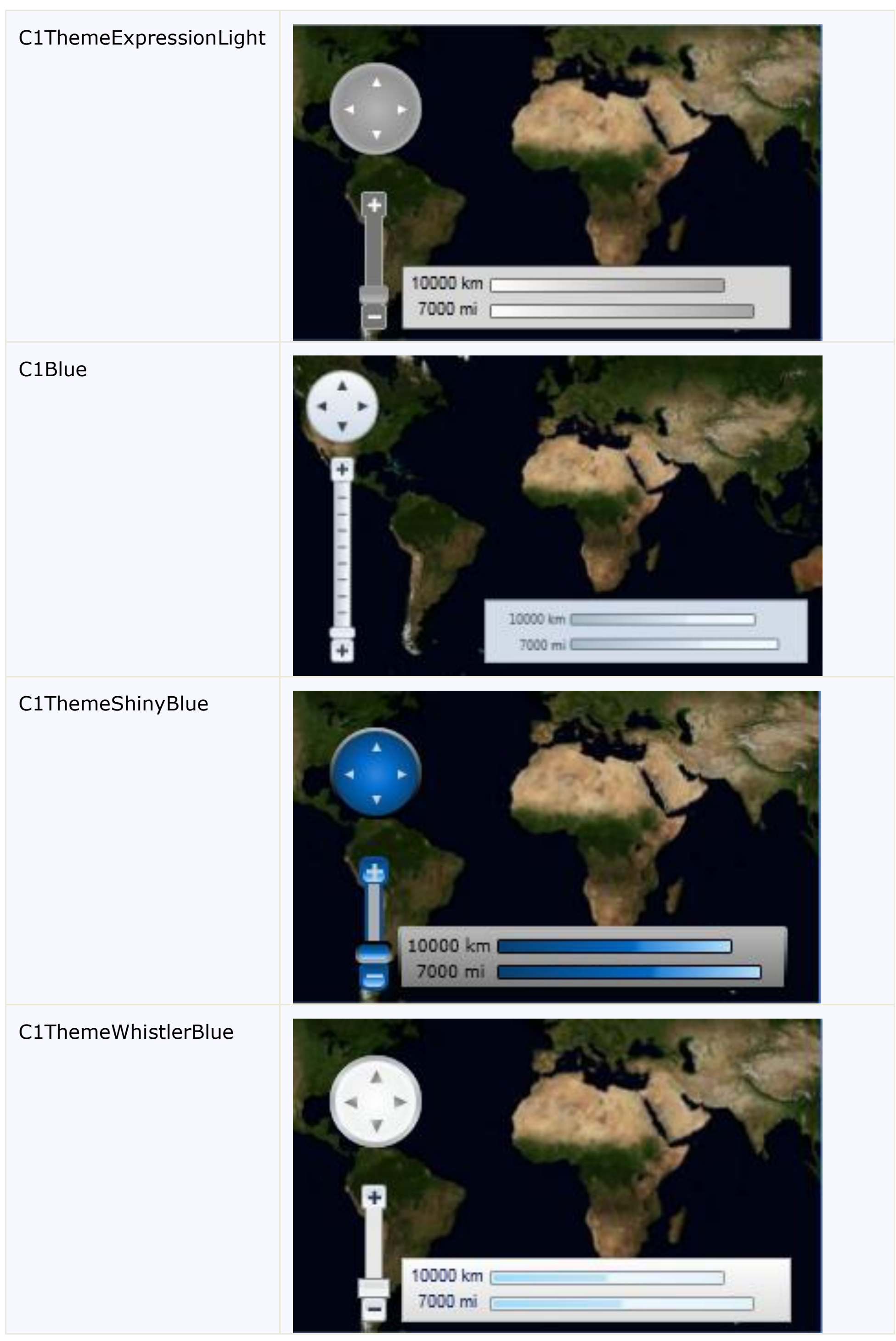
ComponentOne Maps for WPF incorporates several themes that allow you to customize the appearance of your control. When you first add a C1Maps control to the page, it appears similar to the following image:



This is the control's default appearance. You can change this appearance by using one of the built-in themes or by creating your own custom theme. All of the built-in themes are based on WPF Toolkit themes. The built-in themes are described and pictured below; note that in the images below, a row has been selected to show selected styles:



Theme Name	Theme Preview
C1ThemeBureauBlack	
C1ThemeExpressionDark	



To set an element's theme, use the **ApplyTheme** method. First add a reference to the theme assembly to your project, and then set the theme in code, like this:

- Visual Basic

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

```
' Using ApplyTheme
C1Theme.ApplyTheme(LayoutRoot, theme)
```

- C#

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

      //Using ApplyTheme
      C1Theme.ApplyTheme(LayoutRoot, theme);
   }
```

To apply a theme to the entire application, use the **System.Windows.ResourceDictionary.MergedDictionaries** property. First add a reference to the theme assembly to your project, and then set the theme in code, like this:

- Visual Basic

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

   ' Using Merged Dictionaries
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThem
eResources(theme))
```

End Sub

- C#

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

      //Using Merged Dictionaries
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThem
eResources(theme));
   }
```

Note that this method works only when you apply a theme for the first time. If you want to switch to another ComponentOne theme, first remove the previous theme from **Application.Current.Resources.MergedDictionaries**.

Maps for WPF Appearance Properties

ComponentOne Maps for WPF includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Text Properties

The following properties let you customize the appearance of text in the C1Maps control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency

	property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.

Color Properties

The following properties let you customize the colors used in the control itself.

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border.

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1Maps** control.

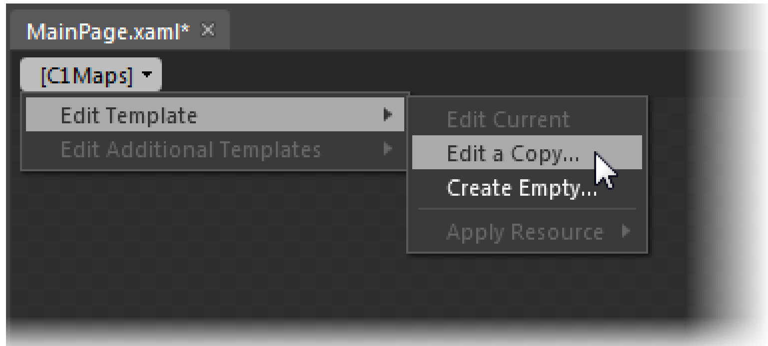
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne Maps for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1Maps control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



> **Note**: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

Maps for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Maps control in general. If you are unfamiliar with the **ComponentOne Maps for WPF** product, please see the **Maps for WPF** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Maps for WPF** product.

Each task-based help topic also assumes that you have created a new WPF project.

Adding a Label

In this topic, you will add a label to a geographic point – the geographic coordinates of Erie, Pennsylvania (USA) - using a C1VectorLayer and a C1VectorPlacemark. For more information on vector layers, see Vector Layer (page 22).

In XAML

Complete the following steps:

1. Add the following XAML between the `<c1:C1Maps>` and `</c1:C1Maps>` tags:

```
<c1:C1VectorLayer>
   <c1:C1VectorPlacemark LabelPosition="Left" GeoPoint="-80.107008,42.16389" StrokeThickness="2" Foreground="#FFEB1212" PinPoint="-80.010866,42.156831" Label="Erie, PA"/>
</c1maps:C1VectorLayer>
```

2. Run the project.

In Code

1. Enter Code view and import the following namespace:
   - Visual Basic

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

   - C#

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

2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

   ```
   ' Create layer and add it to the map
   Dim vl As C1VectorLayer = New C1VectorLayer()
   C1Maps1.Layers.Add(vl)

   'Create a vector placemark and add it to the layer
   Dim vp1 As C1VectorPlacemark = New C1VectorPlacemark()
   vl.Children.Add(vp1)
   ```

```
' Set the placemark to a set of geographical coordinates
vp1.GeoPoint = New Point(-80.107008, 42.16389)

' Set the placemark's label and properties
vp1.Label = "Erie, PA"
vp1.FontSize = 12
vp1.Foreground = New SolidColorBrush(Colors.Red)
vp1.LabelPosition = LabelPosition.Center
```

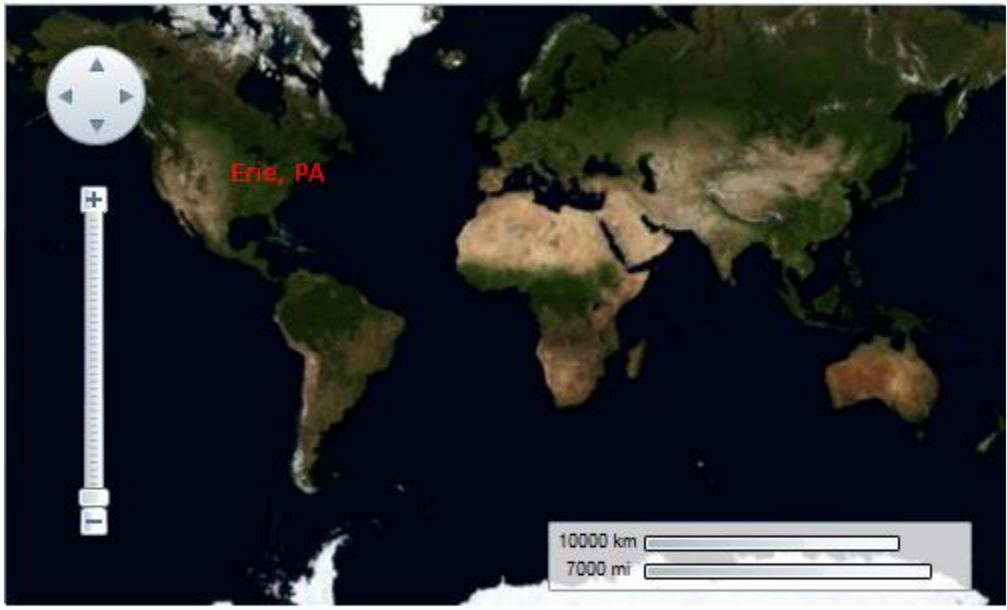
- C#

```
// Create layer and add it to the map
C1VectorLayer vl = new C1VectorLayer();
c1Maps1.Layers.Add(vl);

//Create a vector placemark and add it to the layer
C1VectorPlacemark vp1 = new C1VectorPlacemark();
vl.Children.Add(vp1);

// Set the placemark to a set of geographical coordinates
vp1.GeoPoint = new Point(-80.107008, 42.16389);

// Set the placemark's label and properties
vp1.Label = "Erie, PA";
vp1.FontSize = 12;
vp1.Foreground = new SolidColorBrush(Colors.Red);
vp1.LabelPosition = LabelPosition.Center;
```

3. Run the project.

This Topic Illustrates the Following:

The following image shows a C1Maps control with the geographic coordinates of Erie, Pennsylvania (USA) labeled.



Adding a Polyline

You can connect geographic coordinates with a polyline by adding a C1VectorPolyline to the C1VectorLayer (see Vector Layer (page 22) for more information). In this topic, you will create a 3-point polyline using XAML and code.

In XAML

Complete the following steps:

1. Place the following XAML markup between the `<c1maps:C1Maps>` and `</c1maps:C1Maps>` tags:

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

2. Press F5 to run the project.

In Code

Complete the following steps:

1. Enter Code view and import the following namespace:
   - Visual Basic

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

   - C#

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

2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

   ```
   ' Create layer and add it to the map
   ```

```
Dim C1VectorLayer1 As New C1VectorLayer()
C1Maps1.Layers.Add(C1VectorLayer1)

' Initial track

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

' Create collection and fill it

Dim pcoll As New PointCollection()

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

' Create a polyline and add it to the vector layer as a child

Dim C1VectorPolyline1 As New C1VectorPolyline()
C1VectorLayer1.Children.Add(C1VectorPolyline1)

' Points

C1VectorPolyline1.Points = pcoll

' Appearance

C1VectorPolyline1.Stroke = New SolidColorBrush(Colors.Red)
C1VectorPolyline1.StrokeThickness = 3
```

- C#

```
// Create layer and add it to the map
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
c1Maps1.Layers.Add(C1VectorLayer1);

// Initial track
```

```
Point[] pts = new Point[] { new Point(-80.15,42.12), new Point(-
123.08,39.09),
new Point(-3.90,30.85)};

// Create collection and fill it
PointCollection pcoll = new PointCollection();
foreach( Point pt in pts)
pcoll.Add(pt);

// Create a polyline and add it to the vector layer as a child
C1VectorPolyline C1VectorPolyline1 = new C1VectorPolyline();
v1.Children.Add(C1VectorPolyline1);

  // Points
   C1VectorPolyline1.Points = pcoll;

  // Appearance
  C1VectorPolyline1.Stroke = new SolidColorBrush(Colors.Red);
  C1VectorPolyline1.StrokeThickness = 3;
```

3. Press F5 to run the project.

This Topic Illustrates the Following:

The following image depicts a **C1Maps** control with three geographical coordinates connected by a polyline.

Adding a Polygon

You can connect geographic coordinates with a polygon by adding a C1VectorPolygon to the C1VectorLayer (see Vector Layer (page 22) for more information). In this topic, you will create a 3-point polygon using XAML and code.

In XAML

Complete the following steps:

1. Place the following XAML markup between the `<c1:C1Maps>` and `</c1:C1Maps>` tags:

   ```
   <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. Press F5 to run the project.

In Code

Complete the following steps:

1. In XAML view, add `x:Name="C1Maps1"` to the `<c1:C1Maps>` tag so that the object will have a unique identifier for you to call in code.
2. Enter Code view and import the following namespace:
   - Visual Basic

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

   - C#

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

3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     ' Create layer and add it to the map

     Dim C1VectorLayer1 As New C1VectorLayer()
     C1Maps1.Layers.Add(C1VectorLayer1)

     ' Initial track

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

     ' Create collection and fill it

     Dim pcoll As New PointCollection()

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

```
Next

' Create a polygon and add it to the vector layer as a child

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

' Points

C1VectorPolygon1.Points = pcoll

' Appearance

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

- C#

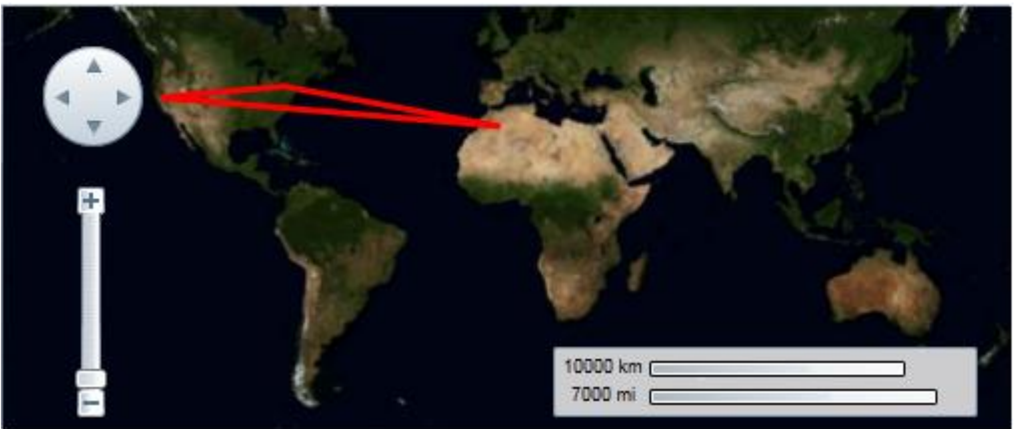
```
// Create layer and add it to the map
C1VectorLayer C1VectorLayer1 = new C1VectorLayer();
c1Maps1.Layers.Add(C1VectorLayer1);

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

// Create collection and fill it
PointCollection pcoll = new PointCollection();
foreach( Point pt in pts)
pcoll.Add(pt);

// Create a polygon and add it to the vector layer as a child
C1VectorPolygon C1VectorPolygon1 = new C1VectorPolygon();
v1.Children.Add(C1VectorPolygon1);

  // Points
   C1VectorPolygon1.Points = pcoll;
```

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

4. Press F5 to run the project.

This Topic Illustrates the Following:

The following image depicts a C1Maps control with three geographical coordinates connected by a polygon.



Displaying Geographic Coordinates on Mouseover

In this topic, you will add code to your project that will return the geographical coordinates of the current mouse position. These geographical coordinates will then be written as a string to the Text property of a **TextBox** control. This task-based help topic assumes that you are working in Visual Studio 2008.

Complete the following steps:

1. In the Toolbox, double-click the **StackPanel** icon to add a StackPanel control to your project.
2. Select the StackPanel control, return to the Toolbox, and double-click the C1Maps icon to add the C1Maps control to the StackPanel control.
3. Select the **StackPanel** control, return to the Toolbox, and double-click the **C1Maps** icon to add the C1Maps control to the **StackPanel** control.
4. Select the **StackPanel** control, return to the Toolbox, and double-click the **TextBox** icon to add the **TextBox** control to the **StackPanel** control.
5. Set the **StackPanel**'s properties as follows:
   - Set the **Width** property to “Auto”.
   - Set the **Height** property to “Auto”.
6. Set the **TextBox** control's **Name** property to "ShowCoordinates".
7. Set the C1Maps control's properties as follows:
   - Set the **Width** property to "350".
   - Set the **Height** property to "250".

8. Select the C1Maps control and then, in the **Properties** window, click the **Events** button .
9. In the **MouseMove** text box, enter "MouseMoveCoordinates" and press ENTER to add the **MouseMoveCoordinates** event handler to your project.
10. Replace the code comment with the following code:
    - 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);
```

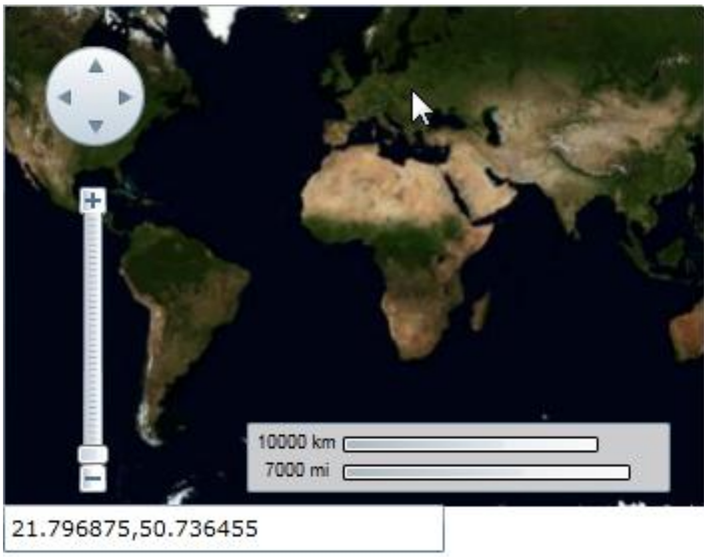
11. Import the following namespace:
    - Visual Basic

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

    - C#

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

12. Press F5 to run the project. Once the project is loaded, run your cursor over the map and observe that geopgraphical coordinates appear in the text box.



Rearranging the Map Tools

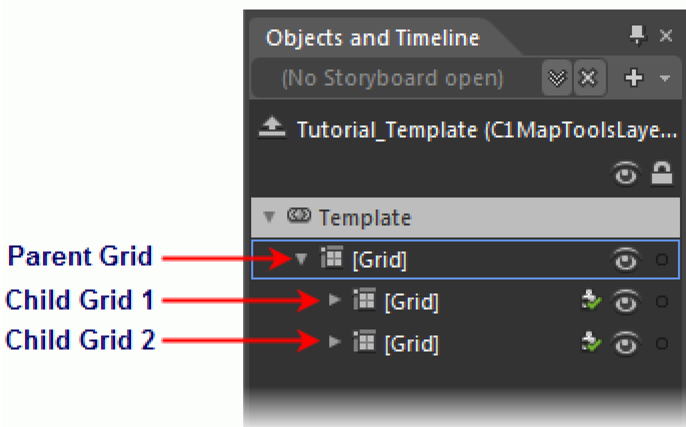
You can modify map tools using the C1MapToolsLayer (see Tool Customization (page 24) for more information) and template. This topic assumes that you've created a Microsoft Expression Blend project. To learn how to use the C1Maps control in Microsoft Expression Blend, see Step 1 of 3: Creating an Application with a C1Maps Control (page 11).

Complete the following steps:

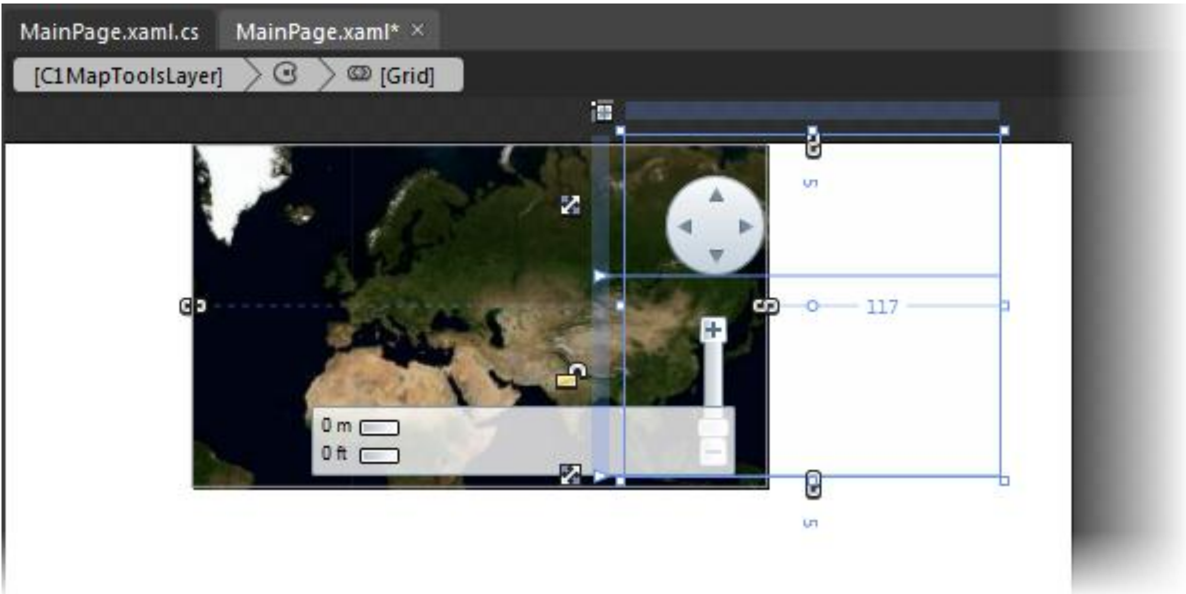
1. Select C1Maps to reveal its list of properties in the **Properties** panel.
2. Clear the **Show Tools** check box. This will hide the default tools.
3. Click the **Layers (Collection)** ellipsis button.

   The **IMapLayer Collection Editor: Layers** dialog box opens.
4. Click **Add another item** to open the **Select Object** dialog box.
5. Select **C1MapToolsLayer** and then press **OK** to close the **Select Object** dialog box.
6. In the **Objects and Timeline** panel, right-click **[C1MapToolsLayer]** and select **Edit Template | Edit a Copy**. Name the template "Tutorial Template" and then press **OK**.

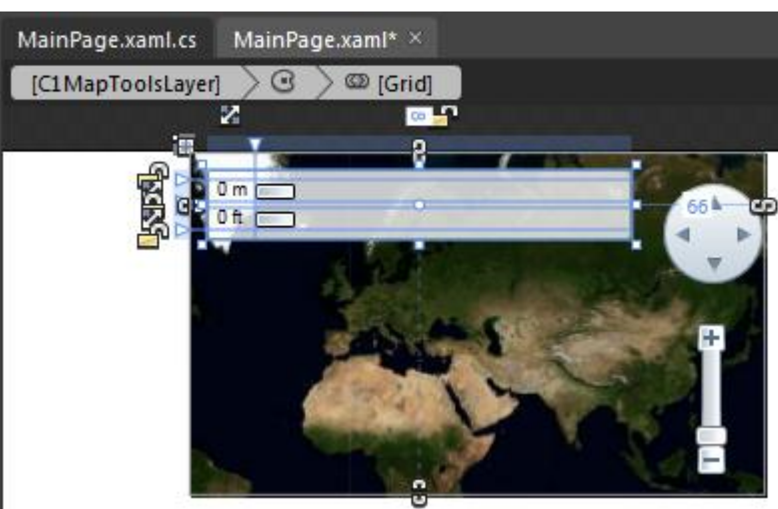
   Your new template is created. Observe that, under the **Objects and Timeline** tab, there is parent grid with two child grids.



7. Click **Child Grid 1** and observe that it takes focus in Design view.
8. In Design view, use your cursor to move the selected grid to the right side of the map so that your project resembles the following:



9. In the **Objects and Timeline** panel, click the **Child Grid 2** to give it focus in Design view.
10. In Design view, use your cursor to move the selected grid to the top-left of the control so that it resembles the following:



11. Press F5 to run the project and observe that the C1Maps control looks as follows:

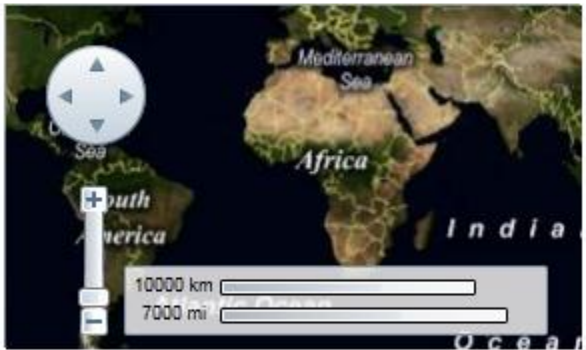
Changing the Map Source

C1Maps can display geographical information from several sources. By default **C1Maps** uses **BingMaps** aerial photographs as the source, but you can change that using the **Source** property, which takes an object of type **MultiScaleTileSource**. By default, this is set to display **Bing Maps**™ (see Legal Requirements (page 17) prior to using this service) aerial photographs, but you can change it to display the road source or hybrid source.

Changing to Road Source

Complete the following steps:

1. Enter Code view and import the following namespace:
   - Visual Basic
     ```
     Imports C1.WPF.C1Maps
     ```
   - C#
     ```
     using C1.WPF.C1Maps;
     ```
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Maps1.Source = new VirtualEarthRoadSource()
     ```
   - C#
     ```
     c1Maps1.Source = new VirtualEarthRoadSource();
     ```
3. Press F5 to run the program and observe that the map presents the road source.



Changing to Hybrid Source

Complete the following steps:

1. Enter Code view and import the following namespace:
   - Visual Basic

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

   - C#

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

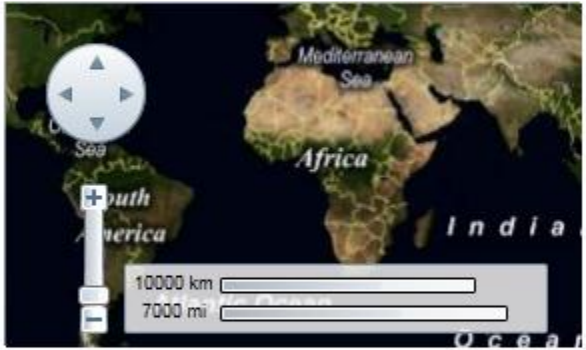
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

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

   - C#

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

3. Press F5 to run the program and observe that the map presents the road source.



For more information about map sources, see C1Maps Concepts and Main Properties (page 18).