ComponentOne

Maps for Silverlight

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 Silverlight Overview 7
Installing Maps for Silverlight 7
Maps for Silverlight Setup Files 7
Using Maps Powered by Esri 8
System Requirements 9
Installing Demonstration Versions 9
Uninstalling Maps for Silverlight 9
End-User License Agreement 9
Licensing FAQs 9
What is Licensing? 9
Maps for Silverlight Licensing 10
Technical Support 12
Redistributable Files 12
About This Documentation 13
Maps for Silverlight Samples 13

Introduction to Silverlight 15
Silverlight Resources 15
Creating a New Silverlight Project 16
Using Templates 18
Data Templates 18
Control Templates 23
Preparing Your Enterprise Environment 26

Theming 27
Available Themes 27
BureauBlack 27
Cosmopolitan 28
ExpressionDark 29
ExpressionLight 29
RainierOrange 30
ShinyBlue 31

WhistlerBlue ... 31
Custom Themes ... 32
Included XAML Files ... 32
C1.Silverlight ... 33
C1.Silverlight.Maps ... 33
C1.Silverlight.Theming.BureauBlack ... 34
C1.Silverlight.Theming.Cosmopolitan ... 34
C1.Silverlight.Theming.ExpressionDark ... 34
C1.Silverlight.Theming.ExpressionLight ... 34
C1.Silverlight.Theming.Office2007 ... 35
C1.Silverlight.Theming.Office2010 ... 35
C1.Silverlight.Theming.RainierOrange ... 35
C1.Silverlight.Theming.ShinyBlue ... 35
C1.Silverlight.Theming.WhistlerBlue ... 36
Implicit and Explicit Styles ... 36
Implicit Styles ... 36
WPF and Silverlight Styling ... 36
Using the ImplicitStyleManager ... 37
Applying Themes to Controls ... 38
Applying Themes to an Application ... 39
ComponentOne ClearStyle Technology ... 41
How ClearStyle Works ... 42
ClearStyle Properties ... 42

Maps for Silverlight Key Features ... 45

Maps for Silverlight Quick Start ... 45
Step 1 of 3: Creating an Application with a C1Maps Control ... 46
Step 2 of 3: Binding to a Data Source ... 46
Step 3 of 3: Running the Project ... 49

Quick XAML Reference ... 51

C1Maps Control Basics ... 53
Legal Requirements ... 53
HTTPS Support ... 53
C1 Concepts and Main Properties ... 54
Items Layering ... 55
Virtualization ... 56
Vector Layer ... 58

Vector Objects 58
Element Visibility 58
KML Import/Export 58
Data Binding 59
Tool Customization 60

Maps for Silverlight Layout and Appearance 61
C1Maps ClearStyle Properties 61
Maps for Silverlight Appearance Properties 61
Text Properties 61
Color Properties 62
Border Properties 62
Size Properties 62
Templates 63
C1Maps Theming 63

Maps for Silverlight Task-Based Help 65
Adding a Label 66
Adding a Polyline 68
Adding a Polygon 71
Displaying Geographic Coordinates on Mouseover 73
Rearranging the Map Tools 75
Changing the Map Source 77
Using C1Maps Themes 79

ComponentOne Maps for Silverlight Overview

ComponentOne Maps™ for Silverlight 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 Silverlight**, visit What's New in Studio for Silverlight.

Getting Started

- C1Maps Control Basics (page 53)

- Quick Start (page 45)

- Task-Based Help (page 65)

Installing Maps for Silverlight

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

Maps for Silverlight Setup Files

The **ComponentOne Maps for Silverlight** installation program will create the following directory: **C:\Program Files\ComponentOne\Studio for Silverlight**. This directory contains the following subdirectories:

Bin	Contains copies of ComponentOne binaries (DLLs, EXEs, design-time assemblies).
Help	Contains documentation for all Studio components and other useful resources including XAML files.

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 Vista/Windows 7 machines:

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

Windows Vista and Windows 7 path: C:\Users\<username>\Documents\ComponentOne Samples\Studio for Silverlight

See the Maps for Silverlight Samples (page 13) topic for more information about each sample.

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 8) or visit the Esri website at http://www.esri.com for additional information.

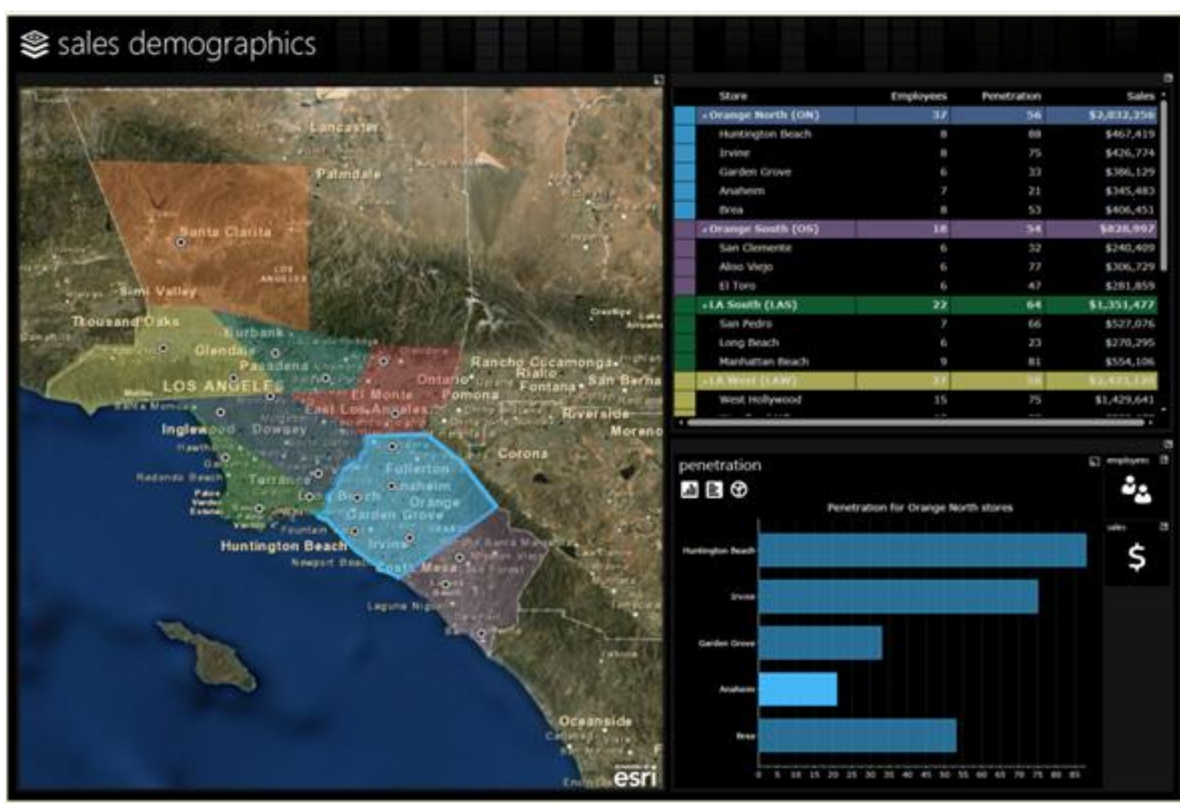
Using Maps Powered by Esri

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.



Esri 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 Silverlight Setup Files (page 7) topic for more information on the Esri files installed with this product.

System Requirements

System requirements for **ComponentOne Maps for Silverlight** include the following:

- Microsoft Silverlight 4.0 or later
- Microsoft Visual Studio 2008 or later

Installing Demonstration Versions

If you wish to try **ComponentOne Maps for Silverlight** 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 the registered version will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling Maps for Silverlight

To uninstall **ComponentOne Maps for Silverlight**:

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

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

The **ComponentOne Maps for Silverlight** product is a commercial product. It is not shareware, freeware, or open source. If you use it in production applications, please purchase a copy from our Web site or from the software reseller of your choice.

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 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.

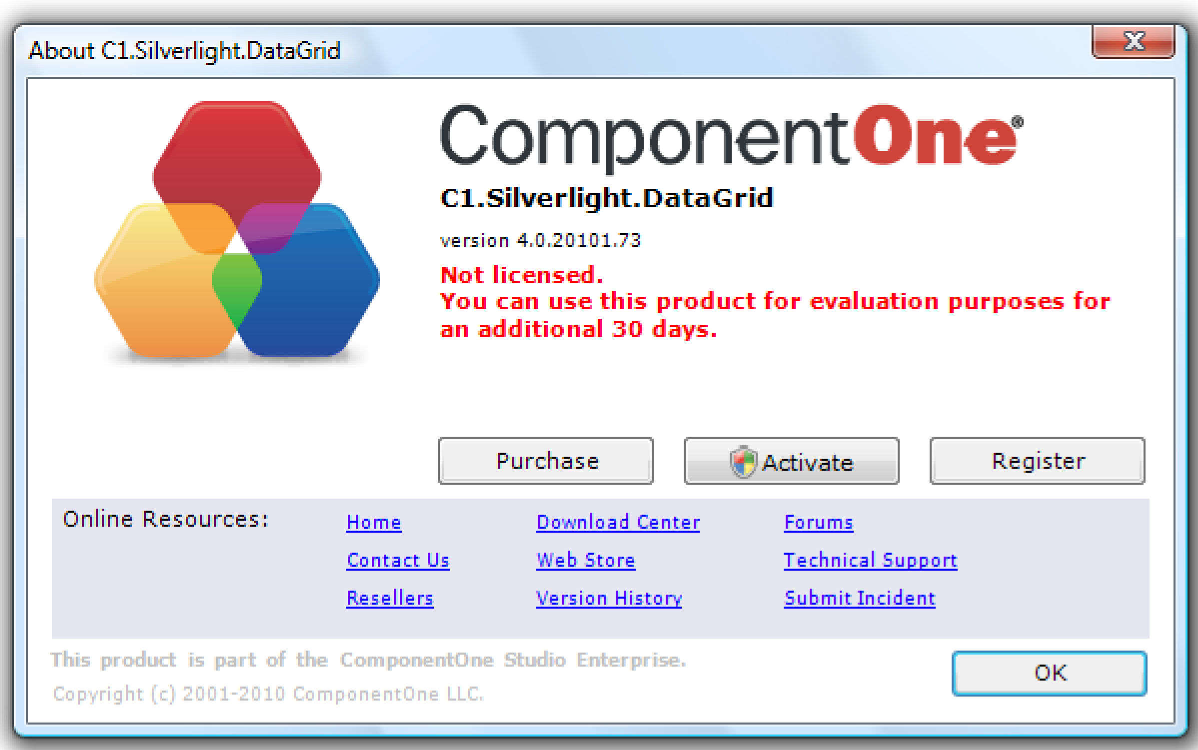
Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software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

Maps for Silverlight Licensing

Licensing for **ComponentOne Maps for Silverlight** is similar to licensing in other ComponentOne products but there are a few differences to note.

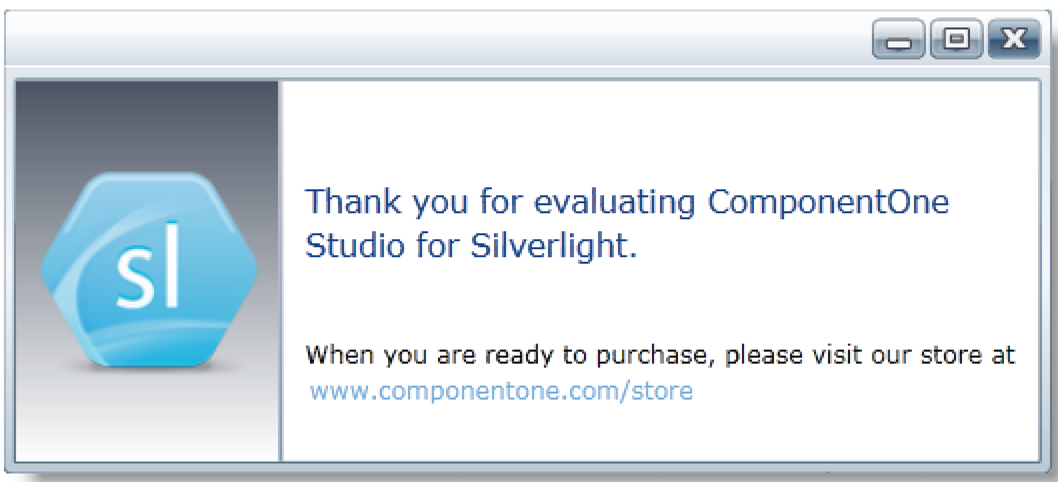
Initially licensing in handled similarly to other ComponentOne products. When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system.

In **ComponentOne Maps for Silverlight**, when a control is dropped on a form, a license nag dialog box appears one time. The nag screen appears similar to the following image:

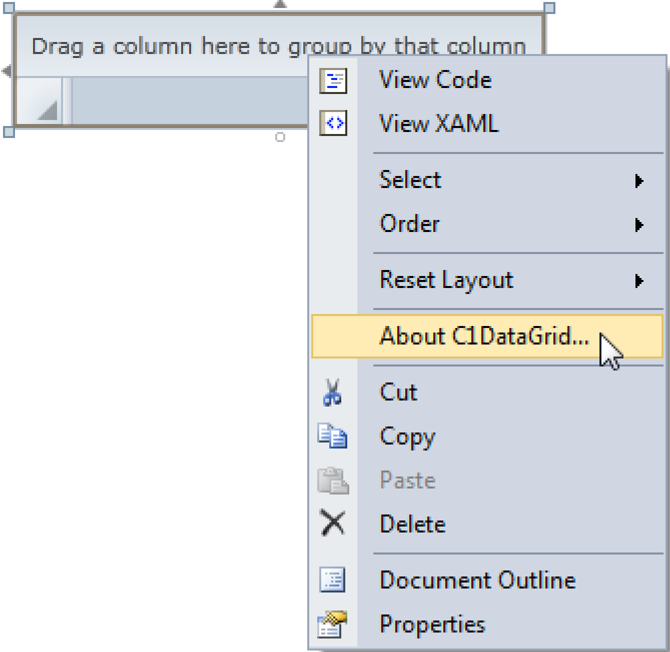


The **About** dialog box displays version information, online resources, and (if the control is unlicensed) buttons to purchase, activate, and register the product.

All ComponentOne products are designed to display licensing information at run time if the product is not licensed. None will throw licensing exceptions and prevent applications from running. Each time an unlicensed Silverlight application is run; end-users will see the following pop-up dialog box:



To stop this message from appearing, enter the product's serial number by clicking the **Activate** button on the **About** dialog box of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box. To open the **About** dialog box, right-click the control and select the **About** option:



Note that when the user modifies any property of a ComponentOne Silverlight control in Visual Studio or Blend, the product will check if a valid license is present. If the product is not currently licensed, an attached property will be added to the control (the **C1NagScreen.Nag** property). Then, when the application executed, the product will check if that property is set, and show a nag screen if the **C1NagScreen.Nag** property is set to **True**. If the user has a valid license the property is not added or is just removed.

One important aspect of this of this process is that the user should manually remove all instances of **c1:C1NagScreen.Nag="true"** in the XAML markup in all files after registering the license (or re-open all the files

that include ComponentOne controls in any of the editors). This will ensure that the nag screen does not appear when the application is run.

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

The **ComponentOne Maps for Silverlight** 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.Silverlight.dll
- C1.Silverlight.Maps.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 Silverlight Samples

If you just installed **ComponentOne Studio for Silverlight**, open Visual Studio 2008 and load the **Samples.sln** solution located in the **C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Silverlight** or **C:\Users\<username>\Documents\ComponentOne Samples\Studio for Silverlight** folder. This solution contains all the samples that ship with this release. Each sample has a readme.txt file that describes it and two projects named as follows:

<SampleName>	Silverlight project (client-side project)
<SampleName>Web	ASP.NET project that hosts the Silverlight project (server-side project)

To run a sample, right-click the **<SampleName>Web** project in the Solution Explorer, select **Set as Startup Project**, and press F5.

Introduction to Silverlight

The following topics detail information about getting started with Silverlight, including Silverlight resources, and general information about templates and deploying Silverlight files.

Silverlight Resources

This help file focuses on **ComponentOne Studio for Silverlight**. For general help on getting started with Silverlight, we recommend the following resources:

- http://www.silverlight.net

  The official Silverlight site, with many links to downloads, samples, tutorials, and more.

- http://silverlight.net/learn/tutorials.aspx

  Silverlight tutorials by Jesse Liberty. Topics covered include:

  - Tutorial 1: Silverlight User Interface Controls
  - Tutorial 2: Data Binding
  - Tutorial 3: Displaying SQL Database Data in a DataGrid using LINQ and WCF
  - Tutorial 4: User Controls
  - Tutorial 5: Styles, Templates and Visual State Manager
  - Tutorial 6: Expression Blend for Developers
  - Tutorial 7: DataBinding & DataTemplates Using Expression Blend
  - Tutorial 8: Multi-page Applications
  - Tutorial 9: ADO.NET DataEntities and WCF Feeding a Silverlight DataGrid
  - Tutorial 10: Hyper-Video

- http://timheuer.com/blog/articles/getting-started-with-silverlight-development.aspx

  Silverlight tutorials by Tim Heuer. Topics covered include:

  - Part 1: Really getting started – the tools you need and getting your first Hello World
  - Part 2: Defining UI Layout – understanding layout and using Blend to help
  - Part 3: Accessing data – how to get data from where
  - Part 4: Binding the data – once you get the data, how can you use it?
  - Part 5: Integrating additional controls – using controls that aren't a part of the core
  - Part 6: Polishing the UI with styles and templates
  - Part 7: Taking the application out-of-browser

- http://weblogs.asp.net/scottgu/pages/silverlight-posts.aspx

  Scott Guthrie's Silverlight Tips, Tricks, Tutorials and Links Page. A useful resource, this page links to several tutorials and samples.

- http://weblogs.asp.net/scottgu/archive/2008/02/22/first-look-at-silverlight-2.aspx

  An excellent eight-part tutorial by Scott Guthrie, covering the following topics:

- Part 1: Creating "Hello World" with Silverlight 2 and VS 2008
    - Part 2: Using Layout Management
    - Part 3: Using Networking to Retrieve Data and Populate a DataGrid
    - Part 4: Using Style Elements to Better Encapsulate Look and Feel
    - Part 5: Using the ListBox and DataBinding to Display List Data
    - Part 6: Using User Controls to Implement Master/Details Scenarios
    - Part 7: Using Templates to Customize Control Look and Feel
    - Part 8: Creating a Digg Desktop Version of our Application using WPF
- http://blogs.msdn.com/corrinab/archive/2008/03/11/silverlight-2-control-skins.aspx

  A practical discussion of skinning Silverlight controls and applications by Corrina Barber.

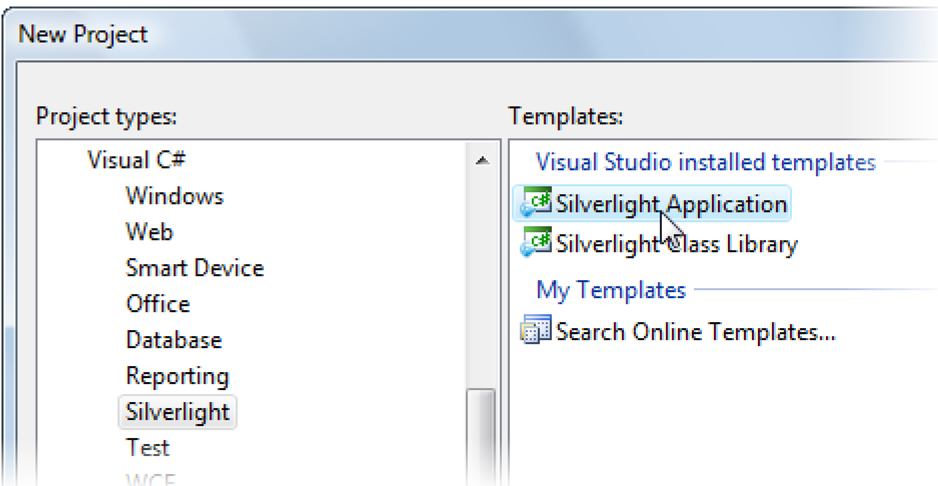
Creating a New Silverlight Project

The following topic details how to create a new Silverlight project in Microsoft Visual Studio 2008 and in Microsoft Expression Blend 3.

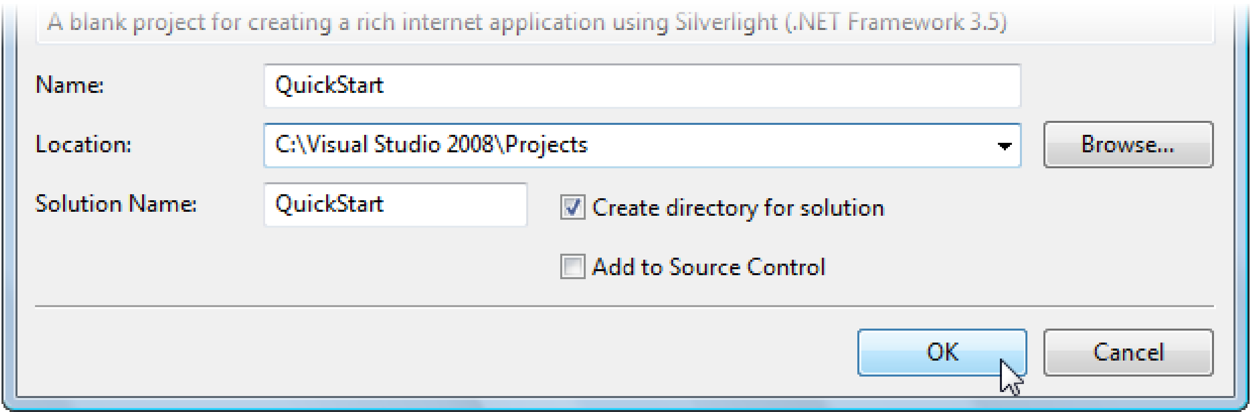
In Visual Studio 2008

Complete the following steps to create a new Silverlight project in Microsoft Visual Studio 2008:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2008.
2. In the **Project types** pane, expand either the **Visual Basic** or **Visual C#** node and select **Silverlight**.
3. Choose **Silverlight Application** in the **Templates** pane.

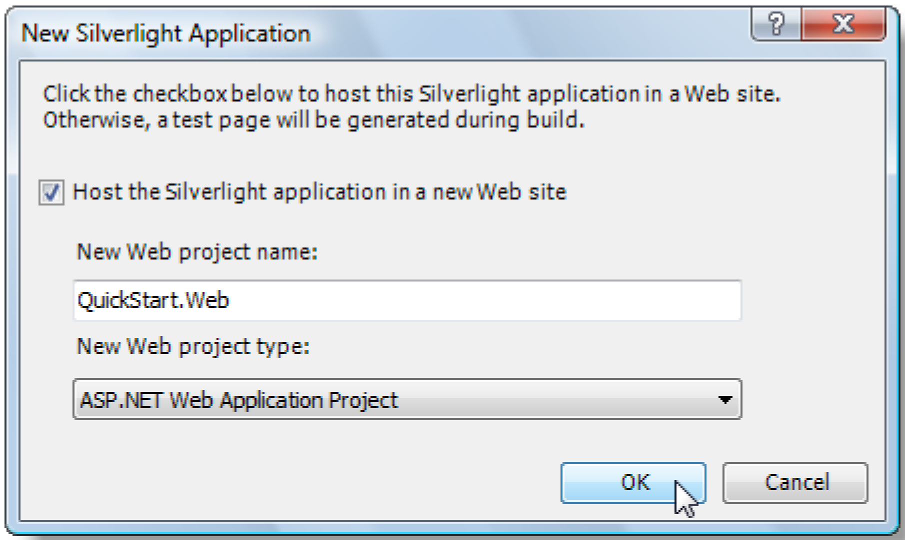


4. Name the project, specify a location for the project, and click **OK**.



Next, Visual Studio will prompt you for the type of hosting you want to use for the new project.

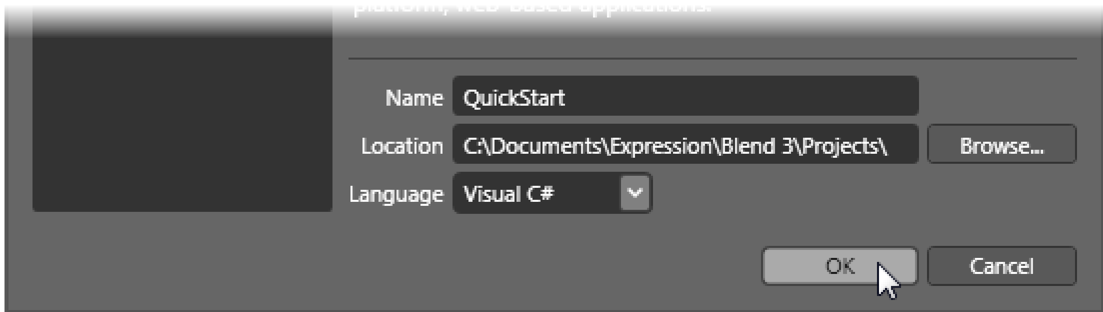
5. In the **NewSilverlight Application** dialog box, select **OK** to accept the default name and options and to create the project.



In Expression Blend 4

Complete the following steps to create a new Silverlight project in Microsoft Expression Blend 4:

1. Select **File | New Project** to open the **New Project** dialog box in Blend 4.
2. In the **Project types** pane, click the **Silverlight** node.
3. In the right pane, choose **Silverlight Application + Website** in the **Templates** pane to create a project with an associated Web site.
4. Name the project, specify a location for the project, choose a language (**Visual C#** or **Visual Basic**), and click **OK**.



Your new project will be created.

The Project

The solution you just created will contain two projects, **YourProject** and **YourProject.Web**:

- **YourProject**: This is the Silverlight application proper. It will produce a XAP file that gets downloaded to the client and runs inside the Silverlight plug-in.
- **YourProject.Web**: This is the host application. It runs on the server and provides support for the Silverlight application.

Using Templates

The previous sections focused on the **ComponentOne Studio for Silverlight** controls. The following topics focus on Data and Control Templates, and how they are applied to Silverlight controls in general (including controls provided by Microsoft). If you are an experienced Silverlight developer, this information may be of no interest to you.

Data Templates

DataTemplates are a powerful feature in Silverlight. They are virtually identical to the **DataTemplates** in WPF, so if you know WPF there's nothing really new about them.

On the other hand, if you have never used WPF and have seen pieces of XAML that contain styles and templates, you may be confused by the concepts and notation. The good news is DataTemplates are very powerful and are not overly complicated. Once you start using them, the concept will make sense in a couple of minutes and you will be on your way. Remember, just reading the tutorial probably won't be enough to fully grasp the concept. After reading, you should play with the projects.

Create the "Templates" Solution

To illustrate the power of DataTemplates, let's create a new Silverlight solution. Call it "Templates". Complete the following steps:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2008.
2. In the **Project types** pane, expand either the **Visual Basic** or **Visual C#** node and select **Silverlight**.
3. Choose **Silverlight Application** in the **Templates** pane.
4. Name the project "Templates", specify a location for the project, and click **OK**.

   Next, Visual Studio will prompt you for the type of hosting you want to use for the new project.
5. In the **New Silverlight Application** dialog box, select **OK** to accept the default name ("Templates.Web") and settings and create the project.
6. Right-click the **Templates** project in the Solution Explorer and select **Add Reference**.

7. In the **Add Reference** dialog box locate and select the C1.Silverlight.dll assembly and click **OK** to add a reference to your project.

   This is required since we will be adding C1.Silverlight controls to the page.

8. Now, open the **MainPage.xaml** file in the **Templates** project and paste in the XAML below:

```
<UserControl x:Class="Templates.MainPage"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:C1_Silverlight="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight">

  <Grid x:Name="LayoutRoot" >
    <Grid.Background>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="#FF7EB9F0"/>
        <GradientStop Color="#FF284259" Offset="1"/>
      </LinearGradientBrush>
    </Grid.Background>

    <!-- Grid layout -->
    <Grid.RowDefinitions>
      <RowDefinition Height="30" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <!-- Page title -->
    <TextBlock Text="Silverlight Templates" Grid.Column="0"
Grid.ColumnSpan="2"
        TextAlignment="Center" FontSize="18" FontWeight="Bold" />

    <!-- ListBox on the left -->
    <StackPanel Grid.Row="1" Margin="5" >
      <TextBlock Text="ListBox Control" />
      <ListBox x:Name="_listBox" />
    </StackPanel>

    <!-- C1ComboBoxes on the right -->
    <StackPanel Grid.Column="2" Grid.Row="1" Margin="5" >
      <TextBlock Text="C1ComboBox Controls" />
      <C1_Silverlight:C1ComboBox x:Name="_cmb1" Margin="0,0,0,5" />
      <C1_Silverlight:C1ComboBox x:Name="_cmb2" Margin="0,0,0,5" />
    </StackPanel>

  </Grid>
</UserControl>
```

   This creates a page with two columns. The left column has a standard **ListBox** control and the right has two **C1ComboBox**es. These are the controls we will populate and style in the following steps.

Populate the Controls

Before we start using templates and styles, let us populate the controls first. To do that, complete the following:

1. Open the **MainPage.xaml.cs** file and paste the following code into the page constructor:

```
public Page()
{
    InitializeComponent();

    // Get list of items
    IEnumerable list = GetItems();

    // Add items to ListBox and in C1ComboBox
    _listBox.ItemsSource = list;
    _cmb1.ItemsSource = list;

    // Show fonts in the other C1ComboBox
    FontFamily[] ff = new FontFamily[]
    {
       new FontFamily("Default font"),
       new FontFamily("Arial"),
       new FontFamily("Courier New"),
       new FontFamily("Times New Roman"),
       new FontFamily("Trebuchet MS"),
       new FontFamily("Verdana")
    };
    _cmb2.ItemsSource = ff;
}
```

The code populates the **ListBox** and both **C1ComboBoxes** by setting their **ItemsSource** property. **ItemsSource** is a standard property present in most controls that support lists of items (**ListBox**, **DataGrid**, **C1ComboBox**, and so on).

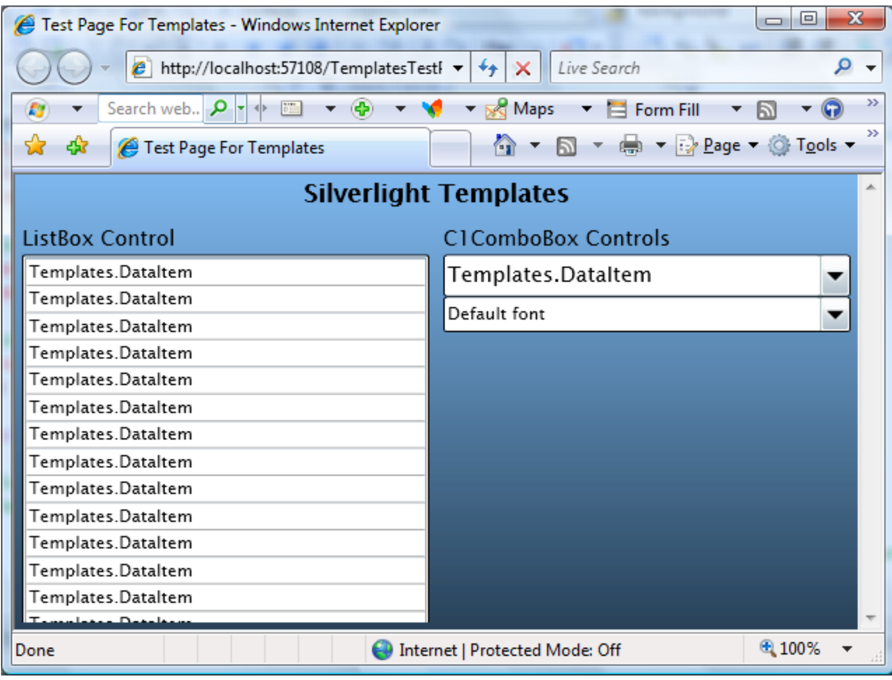
2. Add the following code to implement the **GetItems()** method in the MainPage class:

```
List<DataItem> GetItems()
{
  List<DataItem> members = new List<DataItem>();
  foreach (MemberInfo mi in this.GetType().GetMembers())
  {
    members.Add(new DataItem(mi));
  }
  return members;
}
```

3. Add the definition of the **DataItem** class. to the **MainPage.xaml.cs** file, below the MainPage class definition:

```
public class DataItem
{
    public string ItemName { get; set; }
    public MemberTypes ItemType { get; set; }
    public DataItem(MemberInfo mi)
    {
        ItemName = mi.Name;
        ItemType = mi.MemberType;
    }
}
```

If you run the project now, you will see that the controls are being populated. However, they don't do a very good job of showing the items:



The controls simply convert the **DataItem** objects into strings using their **ToString()** method, which we didn't override and by default returns a string representation of the object type ("Templates.DataItem").

The bottom **C1ComboBox** displays the font family names correctly. That's because the **FontFamily** class implements the **ToString()** method and returns the font family name.

It is easy to provide a **ToString()** implementation that would return a more useful string, containing one or more properties. For example:

```
public override string ToString()
{
    return string.Format("{0} {1}", ItemType, ItemName);
}
```

If you add this method to the **DataItem** class and run the project again, you will see a slightly more satisfying result. But there's only so much you can do with plain strings. To represent complex objects effectively, we need something more. Enter Data Templates!

Defining and Using Data Templates

Data Templates are objects that map regular .NET objects into **UIElement** objects. They are used by controls that contain lists of regular .NET objects to convert these objects into **UIElement** objects that can be displayed to the user.

For example, the Data Template below can be used to map our **DataItem** objects into a **StackPanel** with two **TextBlock** elements that display the **ItemName** and **ItemType** properties of the **DataItem**. This is what the template definition looks like in XAML markup:

```
<UserControl x:Class="Templates.MainPage"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:C1_Silverlight="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight">

  <!-- Data template used to convert DataItem objects into UIElement objects
-->
```

```
  <UserControl.Resources>
    <DataTemplate x:Key="DataItemTemplate" >
      <StackPanel Orientation="Horizontal" Height="30" >
        <TextBlock Text="{Binding ItemType}"
          Margin="5" VerticalAlignment="Bottom" Foreground="Red"
FontSize="10" />
        <TextBlock Text="{Binding ItemName}"
          Margin="5" VerticalAlignment="Bottom" />
      </StackPanel>
    </DataTemplate>
  </UserControl.Resources>

  <!-- Page content (same as before)... -->
```

This template tells Silverlight (or WPF) that in order to represent a source data object, it should do this:

1. Create a **StackPanel** with two **TextBlock**s in it,
2. Bind the **Text** property of the first **TextBlock** to the **ItemType** property of the source data object, and
3. Bind the **Text** property of the second **TextBlock** object to the **ItemName** property of the source object.

That's it. The template does not specify what type of control can use it (any control can, we will use it with the **ListBox** and also with the **C1ComboBox**), and it does not specify the type of object it should expect (any object will do, as long as it has public properties named **ItemType** and **ItemName**).

To use the template, add an **ItemTemplate** attribute to the controls where you want the template to be applied. In our example, we will apply it to the **ListBox** declaration in the **MainPage.xaml** file:

```
<!-- ListBox on the left -->
<StackPanel Grid.Row="1" Margin="5" >
  <TextBlock Text="ListBox Control" />
  <ListBox x:Name="_listBox"
           ItemTemplate="{StaticResource DataItemTemplate}" />
</StackPanel>
```

And also to the top **C1ComboBox**:

```
<!-- C1ComboBox on the right -->
<StackPanel Grid.Column="2" Grid.Row="1" Margin="5" >
  <TextBlock Text="C1ComboBox Controls" />

  <!-- C1ComboBox 1 -->
  <C1_Silverlight:C1ComboBox x:Name="_cmb1" Margin="0,0,0,5"
        ItemTemplate="{StaticResource DataItemTemplate}" />
```

Note that we can now change the appearance of the **DataItem** objects by modifying the template in one place. Any changes will automatically be applied to all objects that use that template, making application maintenance much easier.

Before you run the application again, let's add a template to the second **C1ComboBox** as well. This control contains a list of font families. We can use templates to display each item using the actual font they represent.

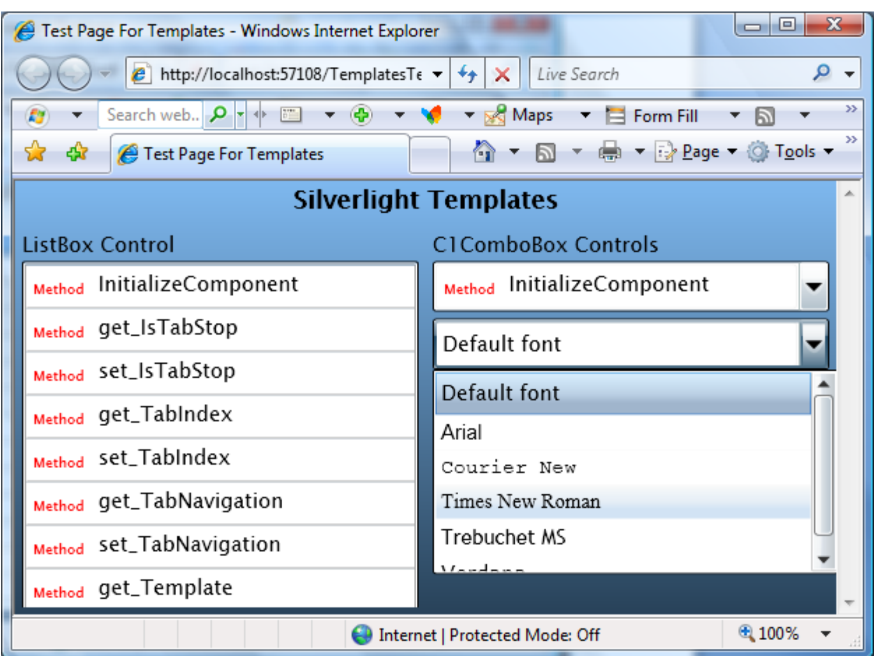
This time, we will not define the template as a resource. It will only be used in one place, so we can insert it inline, as shown below:

```
<!-- C1ComboBox 2 -->
<C1_Silverlight:C1ComboBox x:Name="_cmb2" FontSize="12" Margin="0,0,0,5" >
  <C1_Silverlight:C1ComboBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding}" FontFamily="{Binding}" Margin="4" />
    </DataTemplate>
  </C1_Silverlight:C1ComboBox.ItemTemplate>
</C1_Silverlight:C1ComboBox>
```

Don't let the XAML syntax confuse you. This specifies that in order to create items from data, the control should use a **DataTemplate** that consists of a single **TextBlock** element. The **TextBlock** element should have two of its properties (**Text** and **FontFamily**) bound to the data object itself (as opposed to properties of that object).

In this case, the data object is a **FontFamily** object. Because the template assigns this object to the **Text** property and also to the **FontFamily** property, the **TextBlock** will display the font name and will use the actual font.

If you run the project now, you should see this result:



Note that if you assign a **DataTemplate** to the **C1ComboBox**, it will no longer be able to perform text-related tasks such as auto-search and editing. If you want to re-enable those features, you should provide your own **ItemConverter** that is a standard **TypeConverter**.

Styles and Templates are extremely powerful concepts. We encourage you to play and experiment with this sample. Try modifying the templates to show the data in different ways. The more you experiment, the more comfortable you will feel with these concepts and with the Silverlight/WPF application architecture.

Control Templates

Data Templates allow you to specify how to convert arbitrary data objects into **UIElement** objects that can be displayed to the user. But that's not the only use of templates in Silverlight and WPF. You can also use templates to modify the visual structure of existing **UIElement** objects such as controls.

Most controls have their visual appearance defined by a native XAML resource (typically contained within the assembly that defines the control). This resource specifies a **Style** which assigns values to most of the control's properties, including its **Template** property (which defines the control's internal "visual tree").

For example:

```
<Style TargetType="HyperlinkButton">
  <Setter Property="IsEnabled" Value="true" />
  <Setter Property="IsTabStop" Value="true" />
  <Setter Property="Foreground" Value="#FF417DA5" />
  <Setter Property="Cursor" Value="Hand" />
  <Setter Property="Template">
```

```
    <Setter.Value>
      <ControlTemplate TargetType="HyperlinkButton">
        <Grid x:Name="RootElement" Cursor="{TemplateBinding Cursor}">
          <!-- Focus indicator -->
          <Rectangle x:Name="FocusVisualElement" StrokeDashCap="Round"
...=""/>
          <!-- HyperlinkButton content -->
          <ContentPresenter x:Name="Normal"
               Background="{TemplateBinding Background}"
               Content="{TemplateBinding Content}"
               ContentTemplate="{TemplateBinding ContentTemplate}"...=""/>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

This is a very simplified version of the XAML resource used to specify the **HyperlinkButton** control. It consists of a **Style** that begins by setting the default value of several simple properties, and then assigns a value of type **ControlTemplate** to the control's **Template** property.

The **ControlTemplate** in this case consists of a **Grid** (*RootElement*) that contains a **Rectangle** (*FocusVisualElement*) used to indicate the focused state and a **ContentPresenter** (*Normal*) that represents the content portion of the control (and itself contains another **ContentTemplate** property).

Note the *TemplateBinding* attributes in the XAML. These constructs are used to map properties exposed by the control to properties of the template elements. For example, the **Background** property of the hyperlink control is mapped to the **Background** property of the *Normal* element specified in the template.

Specifying controls this way has some advantages. The complete visual appearance is defined in XAML and can be modified by a professional designer using Expression Blend, without touching the code behind it. In practice, this is not as easy as it sounds, because there are logical relationships between the template and the control implementation.

Recognizing this problem, Silverlight introduced a **TemplatePart** attribute that allows control classes to specify the names and types it expects its templates to contain. In the future, this attribute will be added to WPF as well, and used by designer applications such as Blend to validate templates and ensure they are valid for the target control.

For example, the Microsoft **Button** control contains the following **TemplatePart** attributes:

```
/// <summary>
/// Represents a button control, which reacts to the Click event.
/// </summary>
[TemplatePart(Name = Button.ElementRootName, Type =
typeof(FrameworkElement))]
[TemplatePart(Name = Button.ElementFocusVisualName, Type =
typeof(UIElement))]
[TemplatePart(Name = Button.StateNormalName, Type = typeof(Storyboard))]
[TemplatePart(Name = Button.StateMouseOverName, Type = typeof(Storyboard))]
[TemplatePart(Name = Button.StatePressedName, Type = typeof(Storyboard))]
[TemplatePart(Name = Button.StateDisabledName, Type = typeof(Storyboard))]
public partial class Button : ButtonBase
```

These six template parts constitute a contract between the control implementation and the design specification. They tell the designer that the control implementation expects to find certain elements in the template (defined by their name and type).

Well-behaved controls should degrade gracefully, not crashing if some non-essential elements are missing from the template. For example, if the control can't find a **Storyboard** named *Button.StateMouseOverName* in the template, it should not do anything when the mouse hovers over it.

Well-implemented templates should fulfill the contract and provide all the elements that the control logic supports. Designer applications such as Blend can enforce the contract and warn designers if they try to apply invalid templates to controls.

For the time being, the easiest way to create new templates for existing controls is to start with the original XAML and customize it.

We will not show any actual examples of how to create and use custom control templates here. Instead, we suggest you download the examples developed by Corrina Barber:

http://blogs.msdn.com/corrinab/archive/2008/03/11/silverlight-2-control-skins.aspx

The link contains previews and downloads for three 'skins' (bubbly, red, and flat). Each skin consists of a set of **Style** specifications, similar to the one shown above, which are added to the application's global XAML file (App.xaml). The format is similar to this:

```
<Application xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Styles_Red.App"

<Application.Resources>
  <!-- Button -->
  <Style x:Key="buttonStyle" TargetType="Button">
    <Setter Property="IsEnabled" Value="true" />
    <Setter Property="IsTabStop" Value="true" />
    <Setter Property="Foreground" Value="#FF1E2B33" />
    <Setter Property="Cursor" Value="Hand" />
    <Setter Property="TextAlignment" Value="Center" />
    <!-- A lot more XAML follows… -->
```

Once these styles are defined in the **App.xaml** file, they can be assigned to any controls in the application:

```
<Button Content="Button" Style="{StaticResource buttonStyle}"/>
```

If you are curious, this is what the **Button** control looks like after applying each of the skins defined in the reference above:

This mechanism is extremely powerful. You can change what the controls look like and even the parts used internally to build them.

Unlike data templates, however, control templates are not simple to create and modify. Creating or changing a control template requires not only design talent but also some understanding of how the control works.

It is also a labor-intensive proposition. In addition to their normal appearance, most controls have **Storyboards** that are applied to change their appearance when the mouse hovers over them, when they gain focus, get pressed, get disabled, and so on (see the **C1ComboBox** example above).

Furthermore, all controls in an application should appear consistent. You probably wouldn't want to mix bubbly buttons with regular scrollbars on the same page for example. So each 'skin' will contain styles for many controls.

Some controls are designed with custom templates in mind. For example, the **C1ComboBox** has an **ItemsPanel** property of type **ItemsPanelTemplate**. You can use this property to replace the default drop-down **ListBox** element with any other **UIElement** you like.

For examples of using the **ItemsPanel** property, check the **ControlExplorer** sample installed by default with **ComponentOne Studio for Silverlight**.

Preparing Your Enterprise Environment

Several considerations are important to take into account when planning a corporate deployment of your Silverlight applications in an enterprise environment. For information about these considerations and a description of system requirements and deployment methods as well as the techniques to maintain and support Silverlight after deployment, please see the Silverlight Enterprise Deployment Guide provided by the Microsoft Silverlight team.

The guide helps you to plan and carry out a corporate deployment of Silverlight, and covers:

- Planning the deployment
- Testing deployment strategy
- Deploying Silverlight
- Maintaining Silverlight in your environment

The Silverlight Enterprise Deployment Guide is available for download from the Silverlight whitepapers site: http://silverlight.net/learn/whitepapers.aspx.

Theming

One of the main advantages to using Silverlight is the ability to change the style or template of any control. Controls are "lookless" with fully customizable user interfaces and the ability to use built-in and custom themes. Themes allow you to customize the appearance of controls and take advantage of Silverlight's XAML-based styling. The following topics introduce you to styling Silverlight controls with themes.

You can customize WPF and Microsoft Silverlight controls by creating and modifying control templates and styles. This results in a unique and consistent look for your application.

Templates and styles define the pieces that make up a control and the default behavior of the control, respectively. You can create templates and styles by making copies of the original styles and templates for a control. Modifying templates and styles is an easy way to essentially make new controls in Design view of Microsoft Expression Blend, without having to use code. The following topics provide a detailed comparison of styles and templates to help you decide whether you want to modify the style or template of a control, or both. The topics also discuss the built-in themes available in **ComponentOne Studio for Silverlight**.

Available Themes

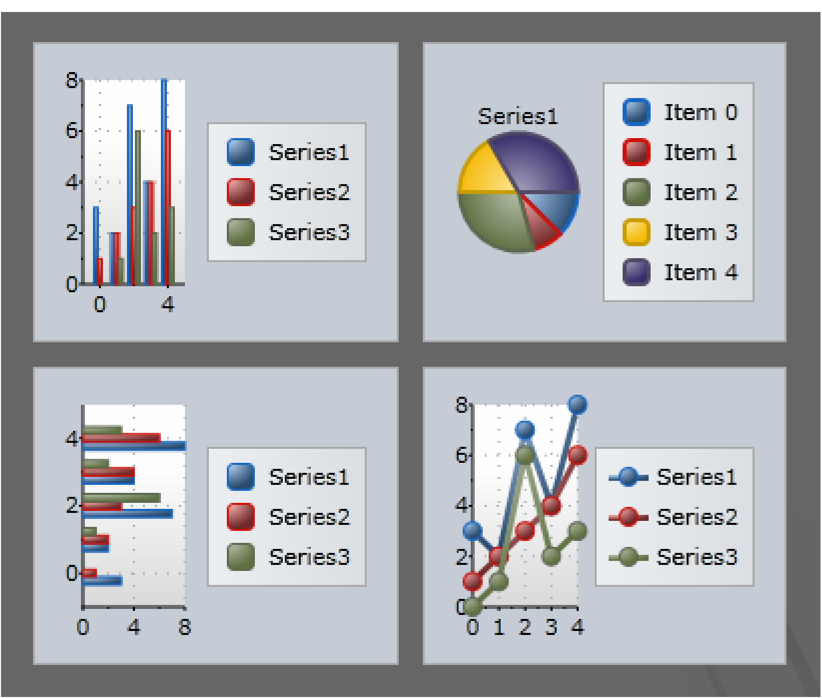
ComponentOne Studio for Silverlight includes several theming options, and several built-in Silverlight Toolkit themes including:

- BureauBlack
- Cosmopolitan
- ExpressionDark
- ExpressionLight
- RainierOrange
- ShinyBlue
- WhistlerBlue

Each of these themes is based on themes in the Silverlight Toolkit and installed in its own assembly in the **Studio for Silverlight** installation directory. The following topics detail each built-in theme.

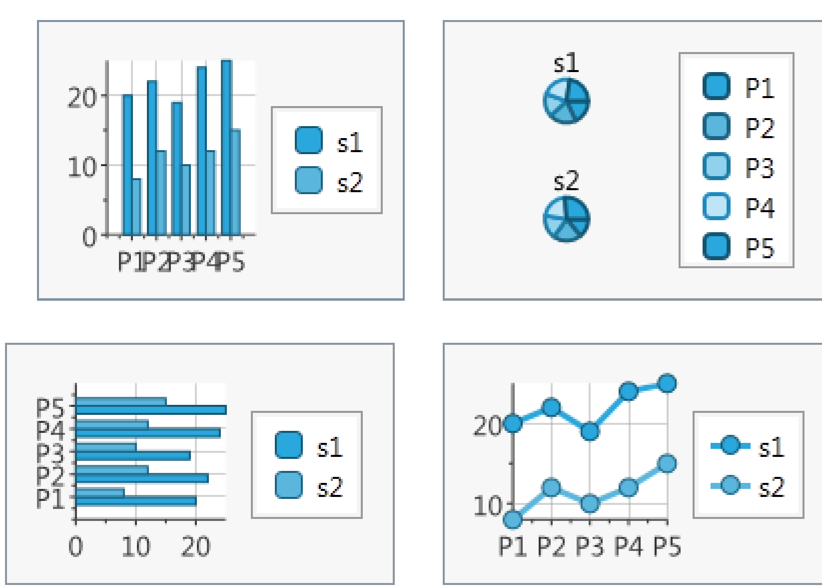
BureauBlack

The BureauBlack theme is a dark colored theme similar to the Microsoft Bureau Black theme included in the Silverlight Toolkit. The BureauBlack theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



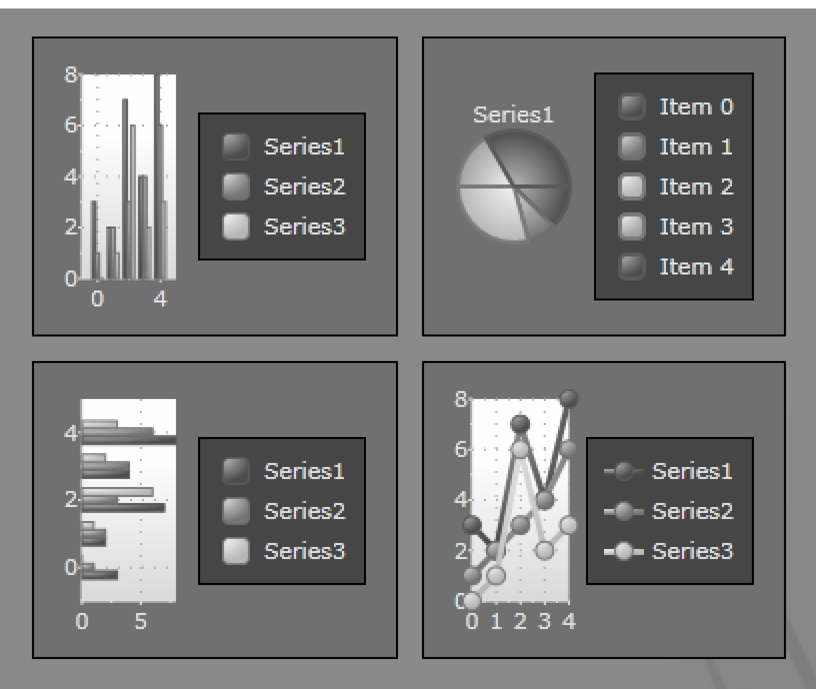
Cosmopolitan

The Cosmopolitan theme is a modern, clean UI theme based on the Microsoft Cosmopolitan theme, which is included in the Silverlight Toolkit. For example, the theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



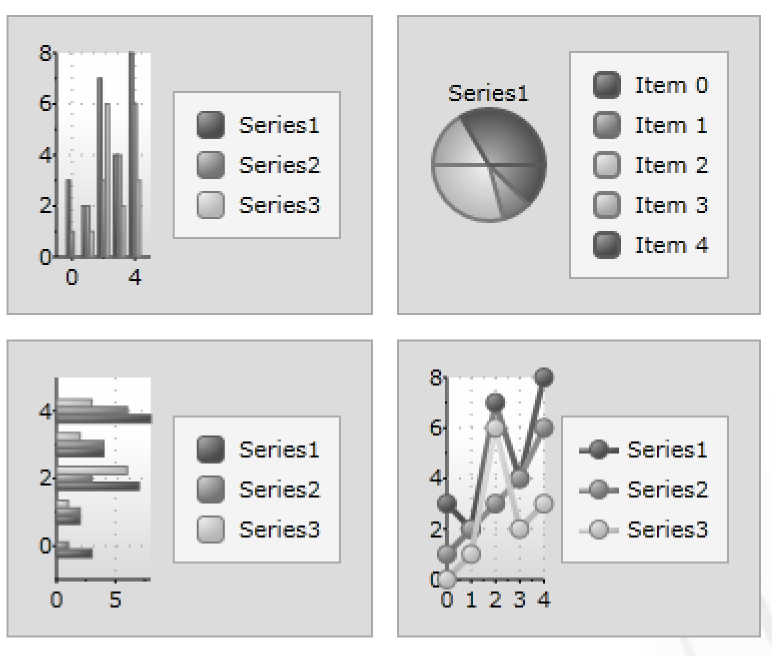
ExpressionDark

The ExpressionDark theme is a grayscale theme based on the Microsoft Expression Dark theme, which is included in the Silverlight Toolkit. For example, the theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



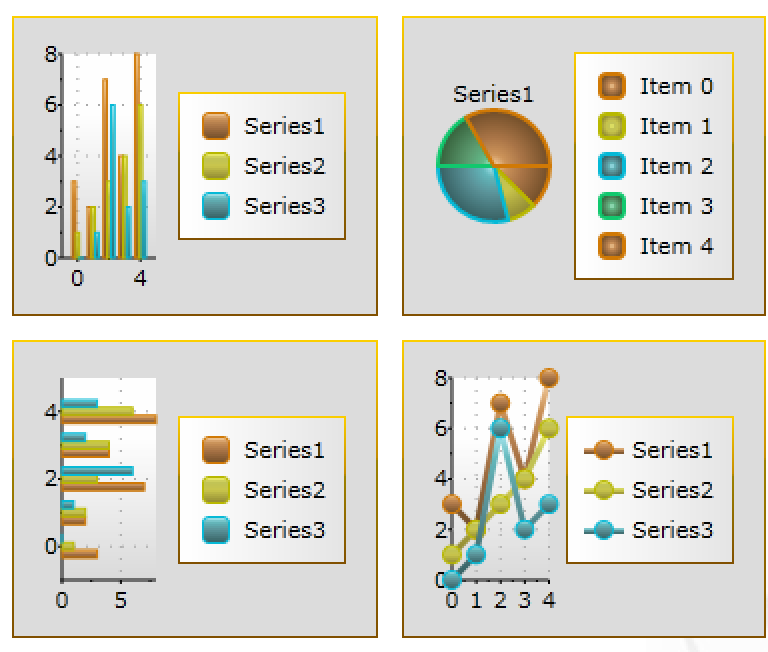
ExpressionLight

The ExpressionLight theme is a grayscale theme based on the Microsoft Expression Light theme, which is included in the Silverlight Toolkit. For example, the theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



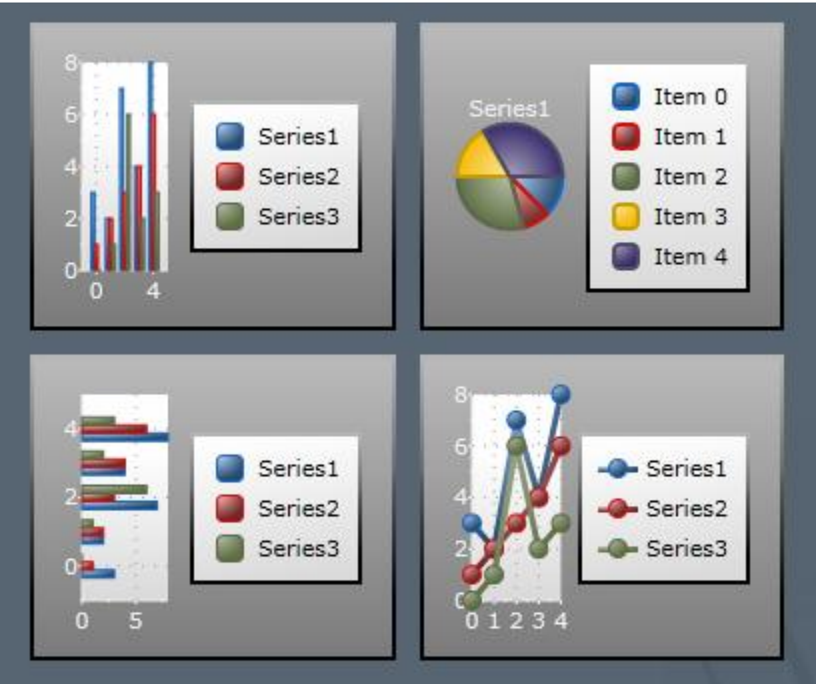
RainierOrange

The RainerOrange theme is an orange-based theme similar to the Microsoft Rainer Orange theme, which is included in the Silverlight Toolkit. The RainerOrange theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



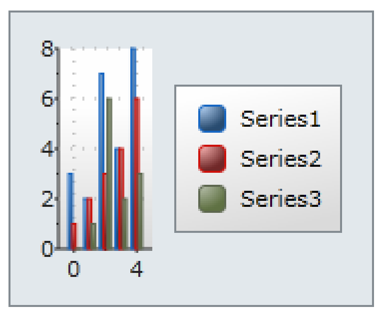
ShinyBlue

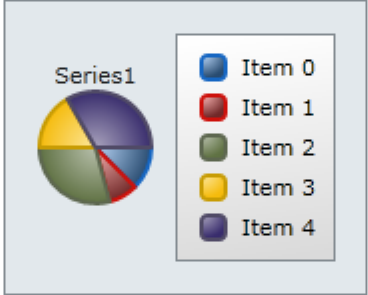
The ShinyBlue theme is a blue-based theme similar to the Microsoft Shiny Blue theme included in the Silverlight Toolkit. The ShinyBlue theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:

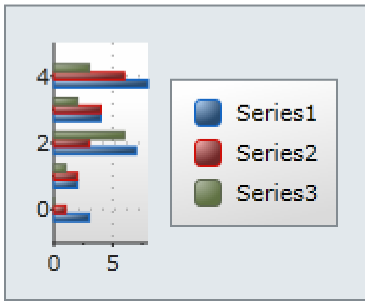


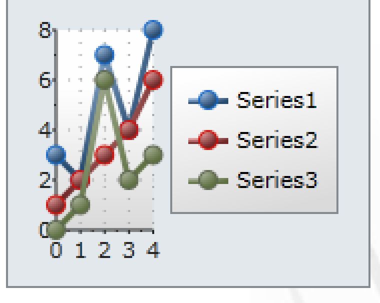
WhistlerBlue

The WhistlerBlue theme is a blue-based theme similar to the Microsoft Whistler Blue theme, which is included in the Silverlight Toolkit. The WhistlerBlue theme appears similar to the following when applied to the **ComponentOne Studio for Silverlight** charting controls:









Custom Themes

In addition to using one of the built-in themes, you can create your own custom theme from scratch or create a custom theme based on an existing built-in theme. See Included XAML Files (page 32) for the included files that you can base a theme on.

Included XAML Files

Several auxiliary XAML elements are installed with **ComponentOne Studio for Silverlight**. These elements include templates and themes and are located in the **Studio for Silverlight** installation directory. You can incorporate these elements into your project to, for example, create your own theme based on the included themes.

By default, these files are located in the **generics.zip** file in the **C:\Program Files\ComponentOne\Studio for Silverlight\Help** folder. Unzip the **generics.zip** file to a folder to see all the XAML files associated with **Studio for Silverlight** controls. In the following topics the included files are listed by assembly with their location folder within the **generics.zip** file noted.

Silverlight 4.0, Silverlight 5.0, and Windows Phone XAML files are named differently. For example, the same version of the generic.xaml file would follow the following naming conventions:

Platform	Name
Silverlight 4.0	generic.xaml
Silverlight 5.0	generic_SL5rd.xaml
Windows Phone	generic.Phone.xaml

The following topics list the Silverlight 4.0 file names, simply add "_SL5rd" or ".Phone" to the file name for the Silverlight 5.0 and Windows Phone files respectively.

C1.Silverlight

The following XAML files can be used to customize items in the **C1.Silverlight** assembly:

Element	Folder	Description
generic.xaml	C1.Silverlight\themes	Specifies the templates for different styles and the initial style of the controls.
Common.xaml	C1.Silverlight\themes	Specifies attributes for common elements in the controls.
C1Button.xaml	C1.Silverlight\themes	Specifies attributes for C1Button.
C1ComboBox.xaml	C1.Silverlight\themes	Specifies attributes for C1ComboBox.
C1DateTimePicker.xaml	C1.Silverlight\themes	Specifies attributes for C1DateTimePicker.
C1DropDown.xaml	C1.Silverlight\themes	Specifies attributes for C1DropDown.
C1FilePicker.xaml	C1.Silverlight\themes	Specifies attributes for C1FilePicker.
C1HeaderedContentControl.xaml	C1.Silverlight\themes	Specifies attributes for C1HeaderedContentControl.
C1LayoutTransformer.xaml	C1.Silverlight\themes	Specifies attributes for C1LayoutTransformer.
C1LoopingList.xaml	C1.Silverlight\themes	Specifies attributes for C1LoopingList.
C1Menu.xaml	C1.Silverlight\themes	Specifies attributes for C1Menu.
C1NumericBox.xaml	C1.Silverlight\themes	Specifies attributes for C1NumericBox.
C1ProgressBar.xaml	C1.Silverlight\themes	Specifies attributes for C1ProgressBar.
C1RangeSlider.xaml	C1.Silverlight\themes	Specifies attributes for C1RangeSlider.
C1ScrollBar.xaml	C1.Silverlight\themes	Specifies attributes for C1ScrollBar.
C1ScrollViewer.xaml	C1.Silverlight\themes	Specifies attributes for C1ScrollViewer.
C1Separator.xaml	C1.Silverlight\themes	Specifies attributes for C1Separator.
C1TabControl.xaml	C1.Silverlight\themes	Specifies attributes for C1TabControl.
C1TextBoxBase.xaml	C1.Silverlight\themes	Specifies attributes for C1TextBoxBase.
C1TextEditableContentControl.xaml	C1.Silverlight\themes	Specifies attributes for C1TextEditableContentControl.
C1ToggleSwitch.xaml	C1.Silverlight\themes	Specifies attributes for C1ToggleSwitch.
C1TreeView.xaml	C1.Silverlight\themes	Specifies attributes for C1TreeView.
C1ValidationDecorator.xaml	C1.Silverlight\themes	Specifies attributes for C1ValidationDecorator.
C1Window.xaml	C1.Silverlight\themes	Specifies attributes for C1Window.

C1.Silverlight.Maps

The following XAML file can be used to customize items in the **C1.Silverlight.Maps** assembly:

Element	Folder	Description
generic.xaml	C1.Silverlight.Maps\themes	Specifies the templates for different styles and the initial style of the controls.
ZoomScrollBar.xaml	C1.Silverlight.Maps\themes	Specifies attributes for the zoom scroll bar.

C1.Silverlight.Theming.BureauBlack

The following XAML files can be used to customize items in the **C1.Silverlight.BureauBlack** assembly:

Element	Folder	Description
BureauBlack.xaml	C1.Silverlight.Theming.BureauBlack	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.Theming.BureauBlack.xaml	C1.Silverlight.Theming.BureauBlack	Specifies the standard Microsoft BureauBlack resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.BureauBlack	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.Cosmopolitan

The following XAML files can be used to customize items in the **C1.Silverlight.Cosmopolitan** assembly:

Element	Folder	Description
Cosmopolitan.xaml/Cosmopolitan_SL5rd.xaml	C1.Silverlight.Theming.Cosmopolitan	Specifies resources and styling elements for each ComponentOne Silverlight control.
Merged.xaml	C1.Silverlight.Theming.Cosmopolitan	Specifies the resources for each ComponentOne WPF control.
Theme.xaml/Theme_SL5rd.xaml	C1.Silverlight.Theming.Cosmopolitan	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ExpressionDark

The following XAML files can be used to customize items in the **C1.Silverlight.ExpressionDark** assembly:

Element	Folder	Description
ExpressionDark.xaml	C1.Silverlight.Theming.ExpressionDark	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.Theming.ExpressionDark.xaml	C1.Silverlight.Theming.ExpressionDark	Specifies the standard Microsoft ExpressionDark resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.ExpressionDark	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ExpressionLight

The following XAML files can be used to customize items in the **C1.Silverlight.ExpressionLight** assembly:

Element	Folder	Description
ExpressionLight.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.Theming.ExpressionLight.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies the standard Microsoft ExpressionLight resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.Office2007

The following XAML files can be used to customize items in the **C1.Silverlight.Office2007** assembly:

Element	Folder	Description
Office2007.xaml	C1.Silverlight.Theming.Office2007	Specifies the standard resources and styling elements.
Office2007Black.xaml	C1.Silverlight.Theming.Office2007	Specifies the standard resources and styling elements.
Office2007Silver.xaml	C1.Silverlight.Theming.Office2007	Specifies the standard resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.Office2007	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.Office2010

The following XAML files can be used to customize items in the **C1.Silverlight.Office2010** assembly:

Element	Folder	Description
Office2010.xaml	C1.Silverlight.Theming.Office2010	Specifies the standard resources and styling elements.
Office2010Black.xaml	C1.Silverlight.Theming.Office2010	Specifies the standard resources and styling elements.
Office2010Silver.xaml	C1.Silverlight.Theming.Office2010	Specifies the standard resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.Office2010	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.RainierOrange

The following XAML files can be used to customize items in the **C1.Silverlight.RainierOrange** assembly:

Element	Folder	Description
RainierOrange.xaml	C1.Silverlight.Theming.RainierOrange	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.RainierOrange	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ShinyBlue

The following XAML files can be used to customize items in the **C1.Silverlight.ShinyBlue** assembly:

Element	Folder	Description
ShinyBlue.xaml	C1.Silverlight.Theming.ShinyBlue	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.ShinyBlue	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.WhistlerBlue

The following XAML files can be used to customize items in the **C1.Silverlight.WhistlerBlue** assembly:

Element	Folder	Description
WhistlerBlue.xaml	C1.Silverlight.Theming.WhistlerBlue	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.WhistlerBlue	Specifies the standard resources and styling elements.

Implicit and Explicit Styles

The following topic detail using implicit and explicit styles and using the **ImplicitStyleManager** which is included in the Silverlight Toolkit. For more information about the Silverlight Toolkit, see CodePlex.

Implicit Styles

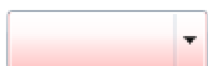
If you're familiar with WPF (Windows Presentation Foundation) you may be used to setting styles implicitly so the application has a uniform appearance – for example, you're used to setting the style for all instances of a particular control in the application's resources. Unfortunately Silverlight does not support implicit styles in the same way that WPF does and you would normally have to indicate the style to use in each instance of the control. This can be tedious to do if you have several controls on a page and that's where the **ImplicitStyleManager** comes in handy. The **ImplicitStyleManager** class is located in the Microsoft.Windows.Controls.Theming namespace (in the Microsoft.Windows.Controls assembly).

WPF and Silverlight Styling

In WPF, you can set styles implicitly. When you set styles implicitly all instances of a particular type can be styled at once. For example, the WPF **C1DropDown** control might be styled with the following markup:

```
<Grid>
    <Grid.Resources>
        <Style TargetType="{x:Type c1:C1DropDown}">
            <Setter Property="Background" Value="Red" />
        </Style>
    </Grid.Resources>
    <c1:C1DropDown Height="30" HorizontalAlignment="Center"
Name="C1DropDown1" VerticalAlignment="Center" Width="100" />
</Grid>
```

This would set the background of the control to be the color red as in the following image:



All **C1DropDown** controls in the grid would also appear red; **C1DropDown** controls outside of the Grid would not appear red. This is what is meant by implicit styles – the style is assigned to all controls of a particular type. Inherited controls would also inherit the style.

Silverlight, however, does not support implicit styles. In Silverlight you could add the style to the Grid's resources similarly:

```
<Grid.Resources>
    <Style x:Key="DropDownStyle" TargetType="c1:C1DropDown">
        <Setter Property="Background" Value="Red" />
    </Style>
</Grid.Resources>
```

But the Silverlight **C1DropDown** control would not be styled unless the style was explicitly set, as in the following example:

```
<c1:C1DropDown Height="30" HorizontalAlignment="Center" Name="C1DropDown1"
VerticalAlignment="Center" Width="100" Style="{StaticResource
DropDownStyle}"/>
```

While this is easy enough to set on one control, if you have several controls it can be tedious to set the style on each one. That's where the **ImplicitStyleManager** comes in. See Using the ImplicitStyleManager (page 37) for more information.

Using the ImplicitStyleManager

The **ImplicitStyleManager** lets you set styles implicitly in Silverlight as you might in WPF. You can find the **ImplicitStyleManger** in the **System.Windows.Controls.Theming.Toolkit.dll** assembly installed with the Silverlight Toolkit.

To use the **ImplicitStyleManager** add a reference in your project to the **System.Windows.Controls.Theming.Toolkit.dll** assembly and add its namespace to the initial **UserControl** tag as in the following markup:

```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight" xmlns:theming="clr-
namespace:System.Windows.Controls.Theming;assembly=System.Windows.Controls.Th
eming.Toolkit" x:Class="C1Theming.MainPage" Width="640" Height="480">
```

Once you've added the reference and namespace you can use the **ImplicitStyleManager** in your application. For example, in the following markup a style is added and implicitly implemented:

```
<Grid x:Name="LayoutRoot" Background="White"
theming:ImplicitStyleManager.ApplyMode="OneTime">
    <Grid.Resources>
        <Style TargetType="c1:C1DropDown">
            <Setter Property="Background" Value="Red" />
        </Style>
    </Grid.Resources>
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <c1:C1DropDown Margin="5" Content="C1DropDown" Height="30"
Width="100"/>
    </StackPanel>
</Grid>
```

Applying Themes to Controls

You can easily customize your application, by applying one of the built-in themes to your ComponentOne Silverlight control. Each of the built-in themes is based on a Silverlight Toolkit theme. For information about each of the built-in themes, see Available Themes (page 27). In this example, you'll add the RainierOrange theme to the **C1DropDown** control on a page.

To apply the theme, complete the following steps:

1. In Visual Studio, select **File | New Project**.
2. In the **New Project** dialog box, select the language in the left pane and in the right-pane select **Silverlight Application**. Enter a **Name** and **Location** for your project and click **OK**.
3. In the **New Silverlight Application** dialog box, leave the default settings and click **OK**.

   A new application will be created and should open with the **MainPage.xaml** file displayed in XAML view.

4. Place the mouse cursor between the <`Grid>` and `</Grid>` tags in XAML view.

   You will add the theme and control to the Grid in the next steps.

5. Navigate to the Visual Studio Toolbox and double-click on the **C1ThemeRanierOrange** icon to declare the theme. The theme's namespace will be added to the page and the theme's tags will be added to the Grid in XAML view. The markup will appear similar to the following:

```
<UserControl xmlns:my="clr-
namespace:C1.Silverlight.Theming.RainierOrange;assembly=C1.Silverlight.
Theming.RainierOrange"  x:Class="C1Silverlight.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
  </Grid>
</UserControl>
```

   Any controls that you add within the theme's tags will now be themed.

6. Place your cursor between the `<my:C1ThemeRanierOrange>` and `</my:C1ThemeRanierOrange>` tags.
7. In the Toolbox, double-click the **C1DropDown** icon to add the control to the project. The C1.Silverlight namespace will be added to the page and the control's tags will be added within the theme's tags in XAML view. The markup will appear similar to the following:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
xmlns:my="clr-
namespace:C1.Silverlight.Theming.RainierOrange;assembly=C1.Silverlight.
Theming.RainierOrange"  x:Class="C1Silverlight.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<my:C1ThemeRainierOrange>
    <c1:C1DropDown Width="100" Height="30"></c1:C1DropDown>
</my:C1ThemeRainierOrange>
```

```
  </Grid>
</UserControl>
```

What You've Accomplished

Run your project and observe that the **C1DropDown** control now appears in the **RainierOrange** theme. Note that you can only set the **Content** property on the theme once, so to theme multiple controls using this method you will need to add a panel, for example a Grid or StackPanel, within the theme and then add multiple controls within the panel.

You can also use the **ImplicitStyleManager** to theme all controls of a particular type. For more information, see Using the ImplicitStyleManager (page 37).

Applying Themes to an Application

The following topic details one method of applying a theme application-wide in Visual Studio. In this topic you'll add a class to your application that initializes a built-in theme. You'll then apply the theme to the MainPage of your application.

To apply the theme, complete the following steps:

1. In Visual Studio, select **File | New Project**.
2. In the **New Project** dialog box, select the language in the left pane and in the right-pane select **Silverlight Application**. Enter a **Name** and **Location** for your project and click **OK**.
3. In the **New Silverlight Application** dialog box, leave the default settings and click **OK**.

   A new application will be created and should open with the **MainPage.xaml** file displayed in XAML view.
4. In the Solution Explorer, right-click the project and choose **Add Reference**.
5. In the **Add Reference** dialog box choose the **C1.Silverlight.Theming** and **C1.Silverlight.Theming.RainierOrange** assemblies and click **OK**.
6. In the Solution Explorer, right-click the project and select **Add | New Item**.
7. In the **Add New Item** dialog box, choose **Class** from the templates list, name the class "MyThemes", and click the **Add** button to create and a new class. The newly created **MyThemes** class will open.
8. Add the following import statements to the top of the class:

   - Visual Basic

   ```
   Imports C1.Silverlight.Theming
   Imports C1.Silverlight.Theming.RainierOrange
   ```

   - C#

   ```
   using C1.Silverlight.Theming;
   using C1.Silverlight.RainierOrange;
   ```
9. Add code to the class so it appears like the following:

   - Visual Basic

   ```
   Public Class MyThemes
       Private _myTheme As C1Theme = Nothing
       Public ReadOnly Property MyTheme() As C1Theme
           Get
               If _myTheme Is Nothing Then
                   _myTheme = New C1ThemeRainierOrange()
               End If
               Return _myTheme
           End Get
       End Property
   ```

```
End Class
```

- C#

```
public class MyThemes
{
    private static C1Theme _myTheme = null;
    public static C1Theme MyTheme
    {
        get
        {
            if (_myTheme == null)
            _myTheme = new C1ThemeRainierOrange();
            return _myTheme;
        }
    }
}
```

10. In the Solution Explorer, double-click the **App.xaml.vb** or **App.xaml.cs** file.
11. Add the following import statement to the top of the file, where *ProjectName* is the name of your application:

- Visual Basic

```
Imports ProjectName
```

- C#

```
using ProjectName;
```

12. Add code to the **Application_Startup** event of the **App.xaml.vb** or **App.xaml.cs** file so it appears like the following:

- Visual Basic

```
Private Sub Application_Startup(ByVal o As Object, ByVal e As
StartupEventArgs) Handles Me.Startup
    Dim MyMainPage As New MainPage()
    Dim themes As New MyThemes
    themes.MyTheme.Apply(MyMainPage)
    Me.RootVisual = MyMainPage
End Sub
```

- C#

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    MainPage MyMainPage = new MainPage();
    MyThemes.MyTheme.Apply(MyMainPage);
    this.RootVisual = MyMainPage;
}
```

Now any control you add to the **MainPage.xaml** file will automatically be themed.

13. Return to the **MainPage.xaml** file and place the mouse cursor between the `<Grid>` and `</Grid>` tags in XAML view.
14. In the Toolbox, double-click the **C1DropDown** icon to add the control to the project.
15. Update the control's markup so it appears like the following:

```
<c1:C1DropDown Width="100" Height="30"></c1:C1DropDown>
```

What You've Accomplished

Run your project and observe that the **C1DropDown** control now appears in the **RainierOrange** theme. To change the theme chosen, now all you would need to do is change the theme in the **MyThemes** class.

For example, to change to the ExpressionDark theme:

1. Add a reference to the C1.Theming.Silverlight.ExpressionDark.dll assembly.
2. Open the **MyThemes** class in your project and add the following import statements to the top of the class:
   - Visual Basic
```
Imports C1.Silverlight.Theming.ExpressionDark
```
   - C#
```
using C1.Silverlight.Theming.ExpressionDark;
```
3. Update code in the class so it appears like the following:
   - Visual Basic
```
Public Class MyThemes
    Private _myTheme As C1Theme = Nothing
    Public ReadOnly Property MyTheme() As C1Theme
        Get
            If _myTheme Is Nothing Then
                _myTheme = New C1ThemeExpressionDark()
            End If
            Return _myTheme
        End Get
    End Property
End Class
```
   - C#
```
public class MyThemes
{
    private static C1Theme _myTheme = null;
    public static C1Theme MyTheme
    {
        get
        {
            if (_myTheme == null)
            _myTheme = new C1ThemeExpressionDark();
            return _myTheme;
        }
    }
}
```

Note that the above steps apply the theme to the **MainPage.xaml** file. To apply the theme to additional pages, you would need to add the following code to each page:

- Visual Basic
```
Dim themes As New MyThemes
themes.MyTheme.Apply(MyMainPage)
```
- C#
```
MyThemes.MyTheme.Apply(LayoutRoot);
```

The theme will then be applied to the page. So, you only have to change one line of code to the class to change the theme, and you only have to add one line of code to each page to apply the theme.

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 Silverlight 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.

ClearStyle Properties

With each release, ComponentOne will be adding ClearStyle functionality to more controls. Currently several Silverlight and WPF controls support ClearStyle. The following table lists all of the ClearStyle-supported Silverlight controls as well as the ClearStyle properties that each supports.

Property	Supported Controls
AlternatingBackground	C1Scheduler
AppointmentForeground	C1Scheduler
AlternatingRowBackground	C1DataGrid
AlternatingRowForeground	C1DataGrid
Background	C1Accordion, C1AccordionItem, C1ColorPicker, C1ComboBox, C1ComboBoxItem, C1ContextMenu, C1CoverFlow, C1DataGrid, C1DateTimePicker, C1Docking, C1DropDown, C1Expander, C1ExpanderButton, C1FilePicker, C1HeaderedContentControl, C1Map, C1MediaPlayer. C1Menu, C1MenuList, C1MenuItem, C1NumericBox, C1Window, C1RangeSlider, C1PropertyGrid, C1Scheduler, C1TabControl, C1TabItem, C1TextBoxBase, C1TimeEditor, C1Toolbar, C1ToolbarGroup, C1ToolbarStrip, C1ToolbarStripItem, C1TreeView, C1TreeViewItem, C1Window
ButtonBackground	C1ComboBox, C1CoverFlow, C1DropDown, C1FilePicker, C1NumericBox, C1TimeEditor, C1ToolbarStrip, C1Window
ButtonForeground	C1ComboBox, C1CoverFlow, C1DropDown, C1FilePicker, C1NumericBox, C1TimeEditor, C1ToolbarStrip, C1Window
CaretBrush	C1ColorPicker, C1ComboBox, C1DateTimePicker, C1NumericBox, C1TextBoxBase, C1TimeEditor
CategoryBackground	C1PropertyGrid

CategoryForeground	C1PropertyGrid
ControlBackground	C1Scheduler
ControlForeground	C1Scheduler
ExpandedBackground	C1AccordionItem, C1Expander, C1ExpanderButton,
FocusBrush	C1ColorPicker, C1ComboBox, C1DataGrid, C1DateTimePicker, C1DropDown, C1Expander, C1ExpanderButton, C1FilePicker, C1MediaPlayer. C1NumericBox, C1Window, C1RangeSlider, C1TextBoxBase, C1TimeEditor, C1Toolbar, C1ToolbarGroup, C1ToolbarStrip, C1ToolbarStripItem
Header	C1Accordion, C1AccordionItem, C1Expander, C1HeaderedContentControl, C1Window
HighlightedBackground	C1ContextMenu, C1Menu, C1MenuList, C1MenuItem
HorizontalGridLinesBrush	C1DataGrid
MouseOverBrush	C1Accordion, C1AccordionItem, C1ColorPicker, C1ComboBox, C1ComboBoxItem, C1CoverFlow, C1DataGrid, C1DateTimePicker, C1Docking, C1DropDown, C1Expander, C1ExpanderButton, C1FilePicker, C1Map, C1MediaPlayer. C1NumericBox, C1RangeSlider, C1PropertyGrid, C1TabControl, C1TabItem, C1TextBoxBase, C1TimeEditor, C1Toolbar, C1ToolbarGroup, C1ToolbarStrip, C1ToolbarStripItem, C1TreeView, C1TreeViewItem, C1Window
OpenedBackground	C1ContextMenu, C1Menu, C1MenuList, C1MenuItem
PressedBrush	C1ColorPicker, C1ComboBox, C1CoverFlow, C1DataGrid, C1DateTimePicker, C1DropDown, C1ExpanderButton, C1FilePicker, C1Map, C1MediaPlayer. C1NumericBox, C1PropertyGrid, C1RangeSlider, C1TextBoxBase, C1TimeEditor, C1Toolbar, C1ToolbarGroup, C1ToolbarStrip, C1ToolbarStripItem, C1Window
RowBackground	C1DataGrid
RowForeground	C1DataGrid
SelectedBackground	C1ComboBox, C1ComboBoxItem, C1DataGrid, C1Scheduler, C1TabControl, C1TabItem, C1TreeView, C1TreeViewItem,
SelectionBackground	C1ColorPicker, C1ComboBox, C1DateTimePicker, C1FilePicker, C1NumericBox, C1TextBoxBase, C1TimeEditor
SelectionForeground	C1ColorPicker, C1ComboBox, C1DateTimePicker, C1FilePicker, C1NumericBox, C1TextBoxBase, C1TimeEditor
TabItemBackground,	C1Docking
TabStripBackground	C1Docking, C1TabControl
TabStripForeground	C1Docking, C1TabControl
TodayBackground	C1Scheduler

Maps for Silverlight Key Features

ComponentOne Maps for Silverlight allows you to create customized, rich applications. Make the most of **Maps for Silverlight** 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.http://www.componentone.com/newimages/Products/Screenshots/StudioSilverlight/C1MapsVectorAsChart.png

- **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 58).

- **Rich Geographical Information**

  Display rich geographical information from various sources, including Bing Map or 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 Silverlight 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 Silverlight 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 58), Virtualization (page 56), and Items Layering (page 55).

- **Silverlight Toolkit Themes Support**

  Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including ExpressionDark, ExpressionLight, WhistlerBlue, RainierOrange, ShinyBlue, and BureauBlack. See C1Maps Theming (page 63).

Maps for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Maps for Silverlight**. 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 Silverlight 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 Silverlight project type in the left pane and, in the right-pane, select **Silverlight 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 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.
5. 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 Silverlight 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")
        }
C1Maps.DataContext = cities
```

- C#

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

5. 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">
</c1>
```

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 Silverlight 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 near **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 Silverlight** quick start. We recommend that you continue to familiarize yourself with the control by visiting the C1Maps Control Basics (page 53) and Maps for Silverlight Task-Based Help (page 65) sections of the Help file.

Quick XAML Reference

This topic is dedicated to providing a quick overview of the XAML used to complete various sks. For more information, see the Maps for Silverlight Task-Based Help (page 65) section.

Item Template

The following example illustrates how to use the maps item template:

```
<c1: x:Name="C1Maps1" FadeInTiles="False" Margin="0,0,235,8" TargetCenter="-65,-25" Center="-58,-25" Zoom="2" Foreground="Aqua">
        <c1:C1Maps.Resources>
        <!—Item template →
           <DataTemplate x:Key="templPts">
             <c1:C1VectorPlacemark
               GeoPoint="{Binding Path=LongLat}" Fill="Aqua" Stroke="Aqua"
```

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

       </c1>
```

Vector Layer Label

The following example illustrates how to create labels using the vector layer:

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

VectorLayer – Polyline

The following example illustrates how to create a polyline (an open line) using the vector layer:

```
<c1:C1Maps>


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

Vector Layer – Polygon

The following example illustrates how to create a polyline (a line that creates a shape) using the vector layer:

```
<c1:C1Maps>


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

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

Theming

The following example illustrates how to apply a theme to the C1Maps control:

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

C1Maps Control Basics

The **C1.Silverlight.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:

- https://www.microsoft.com/maps/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 **C1** control with HTTPS; however, the image tiles *must* come from the same domain as the Silverlight application.

C1 Concepts and Main Properties

This section details basic **C1** concepts and describes the main properties.

Map Source

C1 can display geographical information from several sources. By default, **C1** uses **Microsoft LiveMaps** 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.

> **Note**: **C1Maps** only works in solutions that include a Web site or Web application project. If you use it in a standalone, single-project Silverlight solution, it will not display anything.

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

There are 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 68).
- **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 71).
- **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 66).

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 that are stored in LOD property and 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 (**C1.Zoom** property) in which the element should be displayed.

KML Import/Export

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

KML import is performed by **KmlReader** class that 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 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"},
```

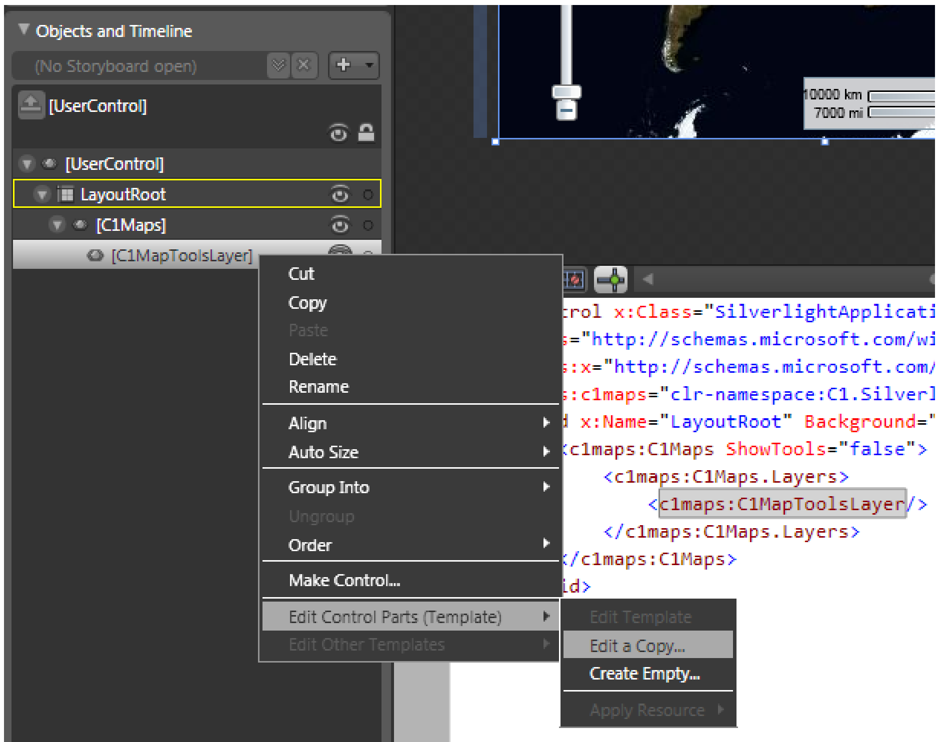
```
    new City(){ LongLat= new Point(10.75,59.91), Name="Oslo"},
    new City(){ LongLat= new Point(12.58,55.67), Name="Copenhagen"}
};

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 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 Silverlight 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 Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

C1Maps ClearStyle Properties

Maps for Silverlight 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 the brush properties of the **C1Maps** control:

Brushes	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 41) topic.

Maps for Silverlight Appearance Properties

ComponentOne Maps for Silverlight 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 **C1** control.

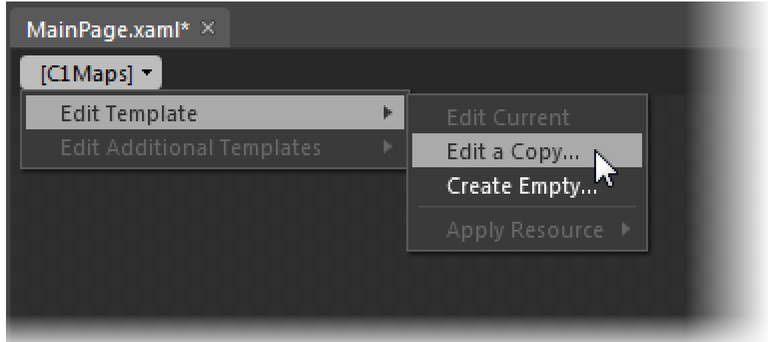
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
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 Silverlight 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 Silverlight applications, you can provide your own UI for data managed by **ComponentOne Maps for Silverlight**. 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.



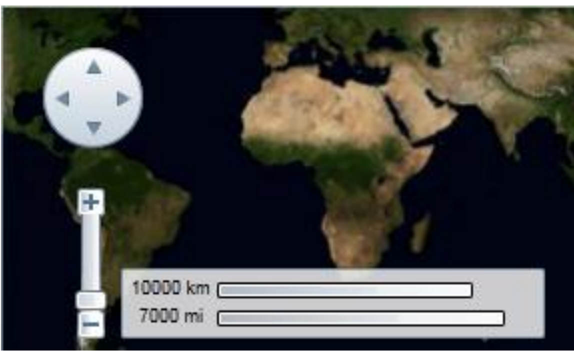
> **Note**: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

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

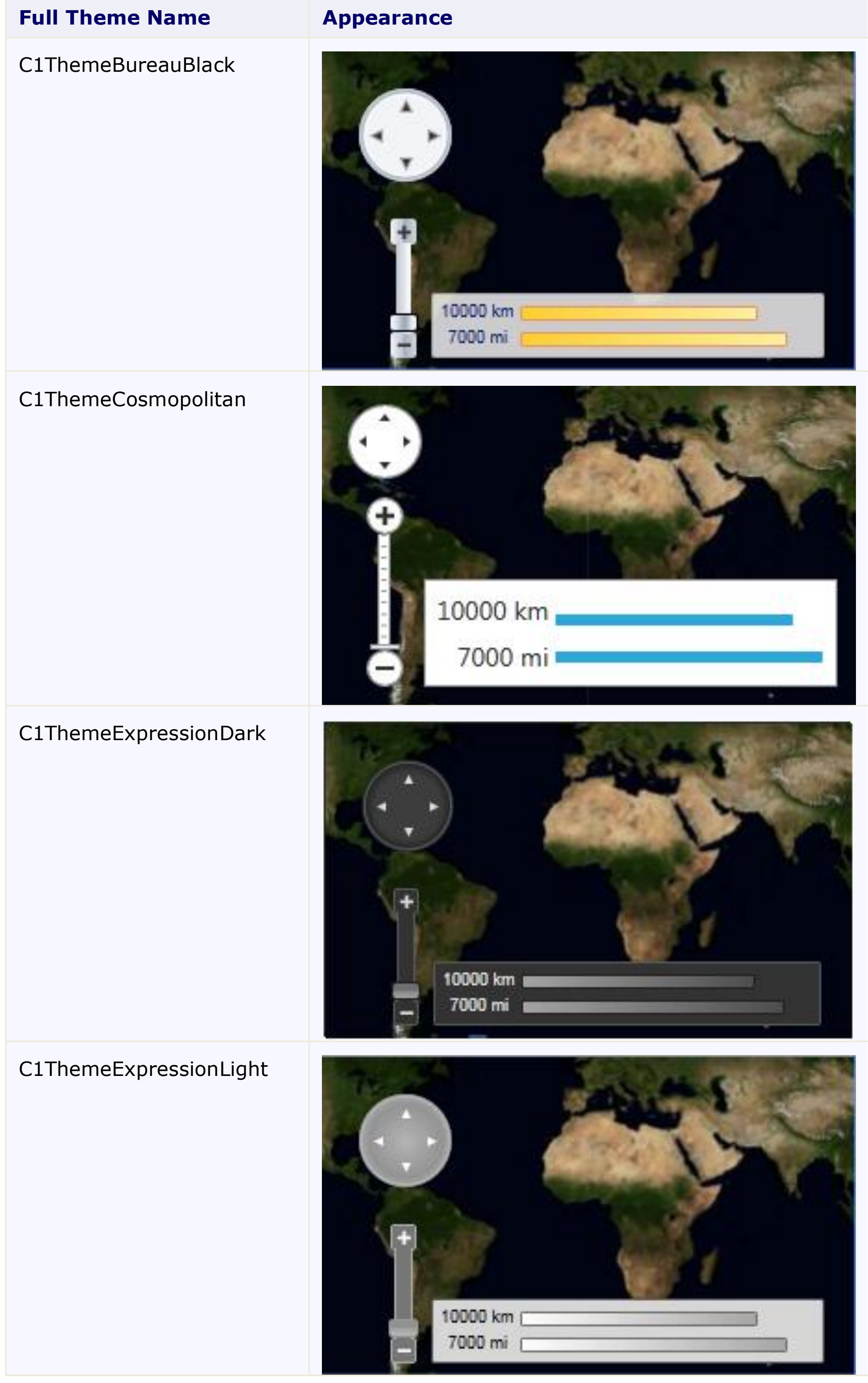
C1Maps Theming

Silverlight themes are a collection of image settings that define the look of a control or controls. The benefit of using themes is that you can apply the theme across several controls in the application, thus providing consistency without having to repeat styling tasks.

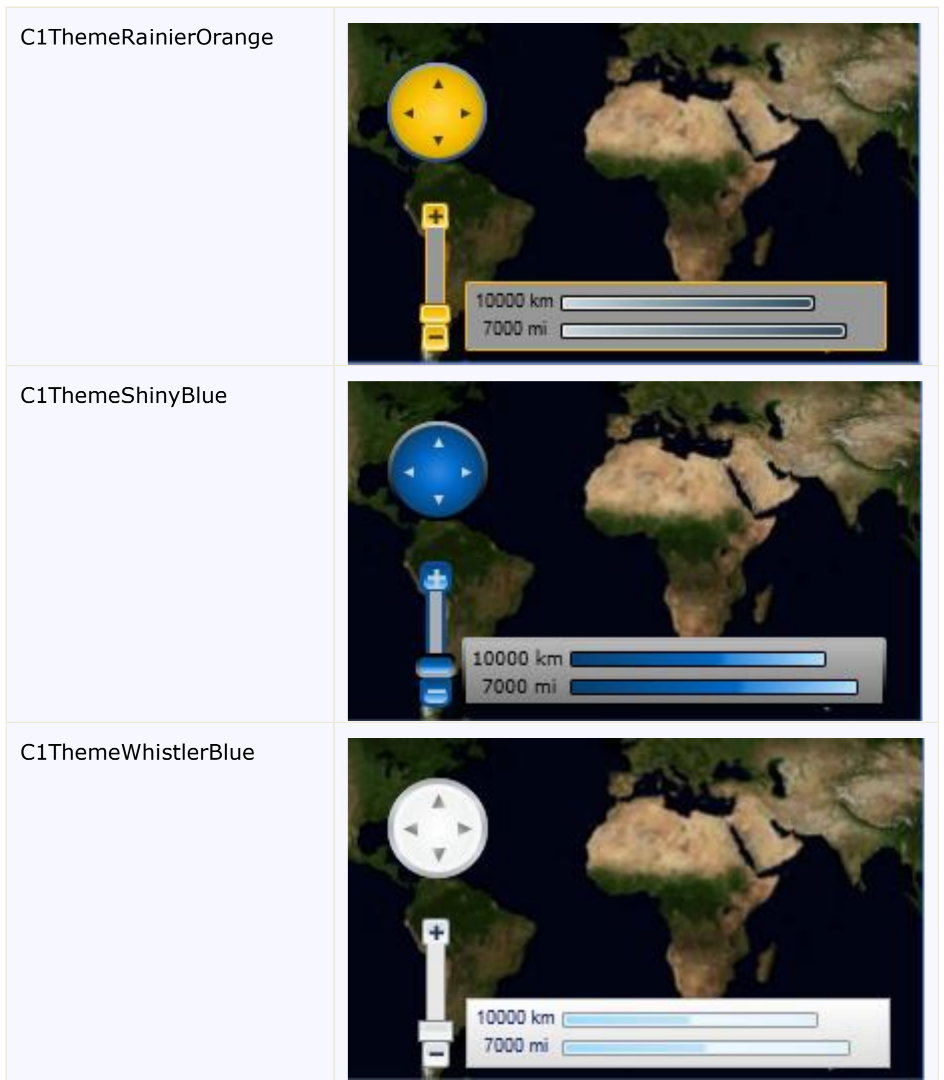
When you add the C1Maps control to your project, it appears with the default theme:



You can theme the C1Maps control with one of our six included Silverlight themes: BureauBlack, ExpressionDark, ExpressionLight, RainierOrange, ShinyBlue, and WhistlerBlue. The table below shows a sample of each theme:



Full Theme Name	Appearance
C1ThemeBureauBlack	
C1ThemeCosmopolitan	
C1ThemeExpressionDark	
C1ThemeExpressionLight	



You can add any of these themes to the C1Maps controls by declaring the theme around the control in markup. For task-based help about adding a theme to the C1Maps control, see Using C1Maps Themes (page 79).

Maps for Silverlight 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 Silverlight** product, please see the **Maps for Silverlight** Quick Start first.

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

Each task-based help topic also assumes that you have created a new Silverlight 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 58).

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"/>
</c1:C1VectorLayer>
```

2. Run the project.

In Code

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.Silverlight.C1Maps
```

   - C#

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

3. 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
```

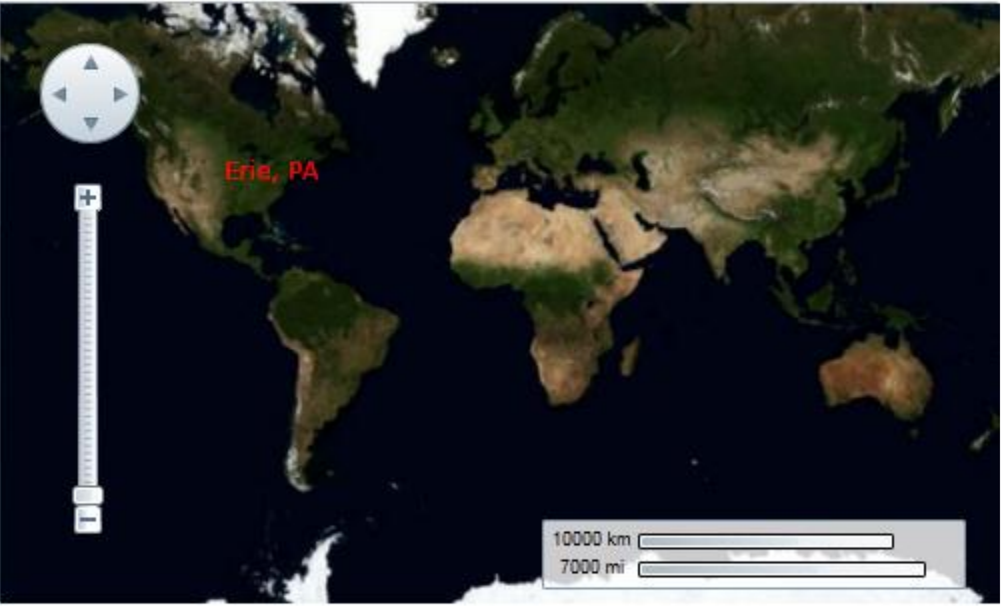
- C#

```
// 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;
```

4. 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 58) 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 <c1:C1Maps> and </c1: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. 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.Silverlight.C1Maps
     ```

   - C#

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

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 polyline.

Adding a Polygon

You can connect geographic coordinates with a polygon by adding a C1VectorPolygon to the C1VectorLayer (see Vector Layer (page 58) 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.Silverlight.C1Maps
   ```

   - C#

   ```
   using C1.Silverlight.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:

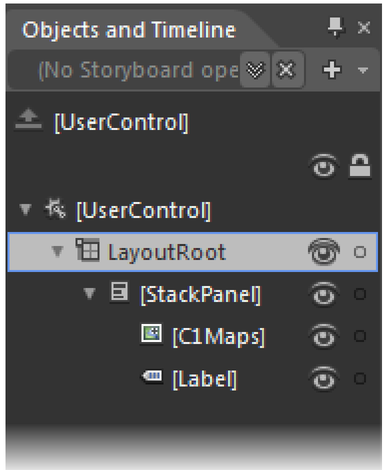
The following image depicts a C1Maps control with three geographical coordinates connected by a polygon.

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.

Complete the following steps:

1. Add a **StackPanel**, a **TextBox** control, and a C1Maps control to your project.
2. In the **Objects and Timeline** panel, rearrange the controls so they appears as follows:



3. Set the **StackPanel**'s properties as follows:
   - Locate the **Width** property and click its glyph to set the **Width** property to **Auto**.
   - Locate the **Height** property and click its glyph to set the **Height** property to **Auto**.
4. Set the **TextBox** control's **Name** property to "ShowCoordinates".
5. Set the C1Maps control's properties as follows:
   - Set the **Width** property to "350".
   - Set the **Height** property to "250".
6. Select the C1Maps control and then, in the **Properties** panel, click the **Events** button .
7. In the **MouseMove** text box, enter "MouseMoveCoordinates" and press ENTER to add the **MouseMoveCoordinates** event handler to your project.
8. 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);
```

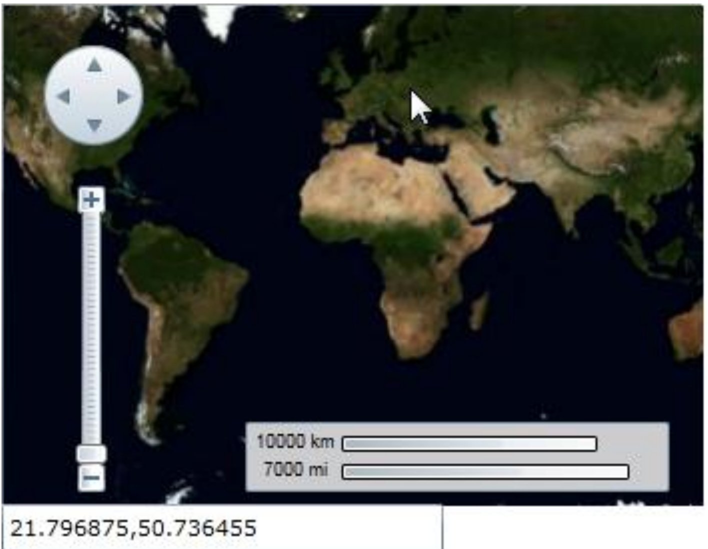
9. Import the following namespace:
   - Visual Basic

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

   - C#

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

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



Rearranging the Map Tools

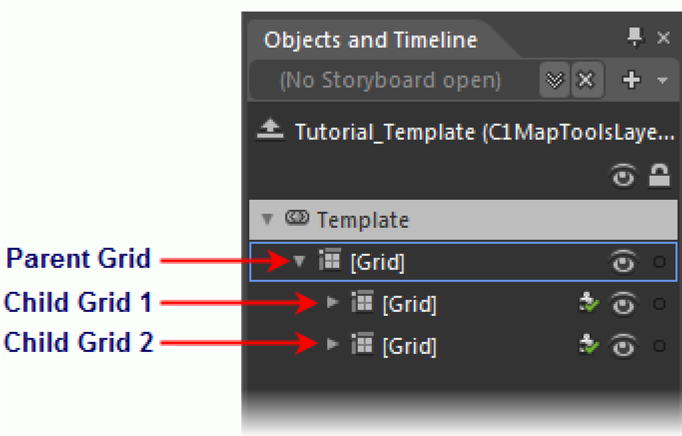
You can modify map tools using the C1MapToolsLayer (see Tool Customization (page 60) for more information) and template.

Complete the following steps:

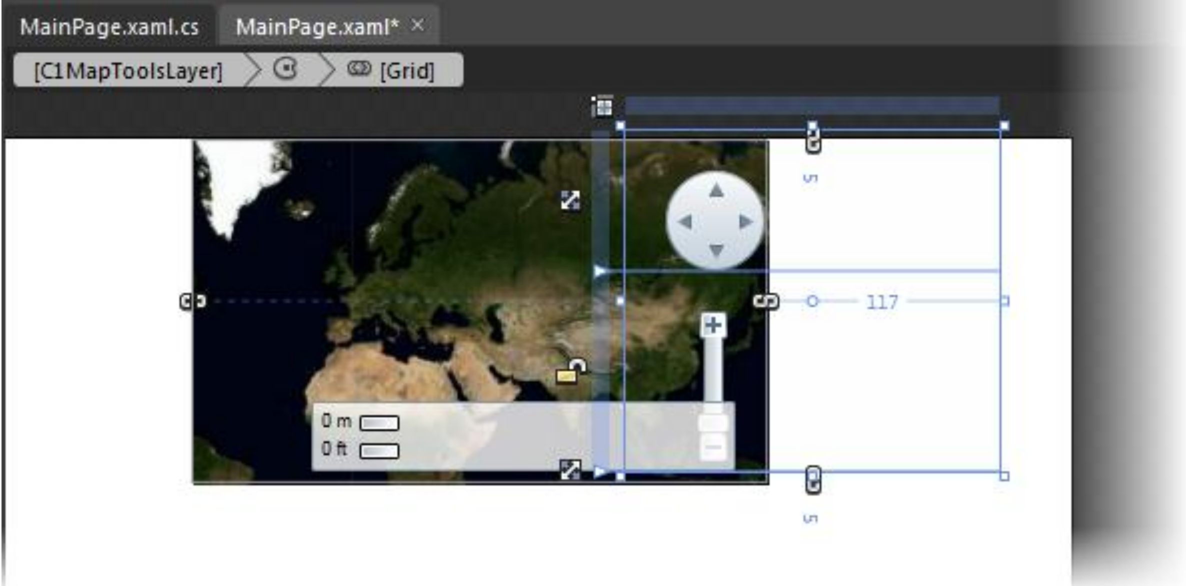
1. Select C1Maps to reveal its list of properties in the **Properties** panel.
2. Clear the **Show Tools** check box. This will hide the default tools.
3. Click the **Layers (Collection)** ellipsis button.

   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**.

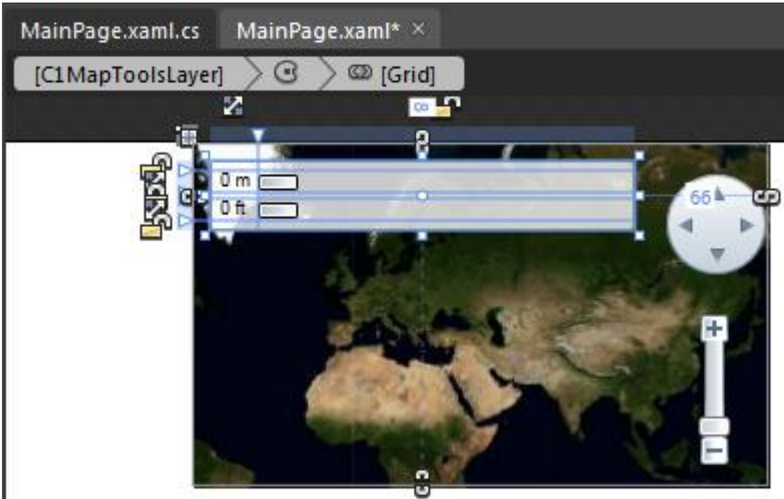
   Your new template is created. Observe that, under the **Objects and Timeline** tab, there is parent grid with two child grids.



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 **Microsoft LiveMaps** 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 53) 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. 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.Silverlight.C1Maps
     ```

   - C#

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

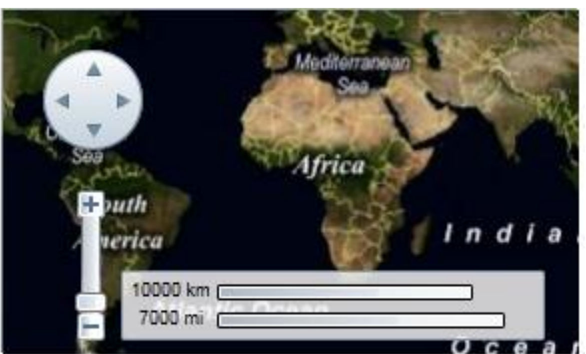
3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

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

   - C#

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

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



Changing to Hybrid Source

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.Silverlight.C1Maps
     ```

   - C#

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

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

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

   - C#

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

4. 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 54).

Using C1Maps Themes

The C1Maps control comes equipped with a light blue default theme, but you can also apply six themes (see C1Maps Theming (page 63)) to the control. In this topic, you will change the C1Maps control's theme to **C1ThemeRainierOrange**.

In Blend

Complete the Following steps:

1. Click the **Assets** tab.
2. In the search bar, enter "C1ThemeRainierOrange".

   The **C1ThemeRainierOrange** icon appears.
3. Double-click the **C1ThemeRainierOrange** icon to add it to your project.
4. In the search bar, enter "C1Maps" to search for the C1Maps control.
5. Double-click the C1Maps icon to add the C1Maps control to your project.
6. In the **Objects and Timeline** panel, select **[C1Maps]** and use a drag-and-drop operation to place it under **[C1ThemeRainierOrange]**.
7. Run the project.

In Visual Studio

Complete the following steps:

1. Open the **.xaml** page in Visual Studio.
2. Place your cursor between the <Grid></Grid> tags.
3. In the **Tools** panel, double-click the **C1ThemeRainierOrange** icon to declare the theme. Its tags will appear as follows:

```
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```

4. Place your cursor between the <my:C1ThemeRainierOrange> and </my:C1ThemeRainierOrange> tags.

5. In the **Tools** panel, double-click the C1Maps icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRainierOrange>` tags, causing the markup to resemble the following:

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

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1Maps control with the **C1ThemeRainierOrange** theme.