ComponentOne

Chart3D for Silverlight

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)

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

Table of Contents

ComponentOne Chart 3D for Silverlight Overview

Graph your data in three dimensions. With **ComponentOne Chart3D™** for Silverlight you can create professional looking 3D surface charts with options for contour levels and zones. Rotate the chart to any angle, show a chart legend and more.

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

Getting Started

Get started with the following topics:

- Key Features (page 19)

- Chart3D for Silverlight Quick Start (page 21)

- Chart Types (page 25)

Installing Chart3D for Silverlight

The following sections provide helpful information on installing **ComponentOne Chart3D for Silverlight**.

Chart3D for Silverlight Setup Files

The **ComponentOne Chart3D 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

System Requirements

System requirements for **ComponentOne Chart3D for Silverlight** include the following:

1. Microsoft Silverlight 4.0 or later
2. Microsoft Visual Studio 2008 or later

Installing Demonstration Versions

If you wish to try **ComponentOne Chart3D 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 Chart3D for Silverlight

To uninstall **ComponentOne Chart3D 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 4.0** 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 Studio 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.

Studio for Silverlight Licensing

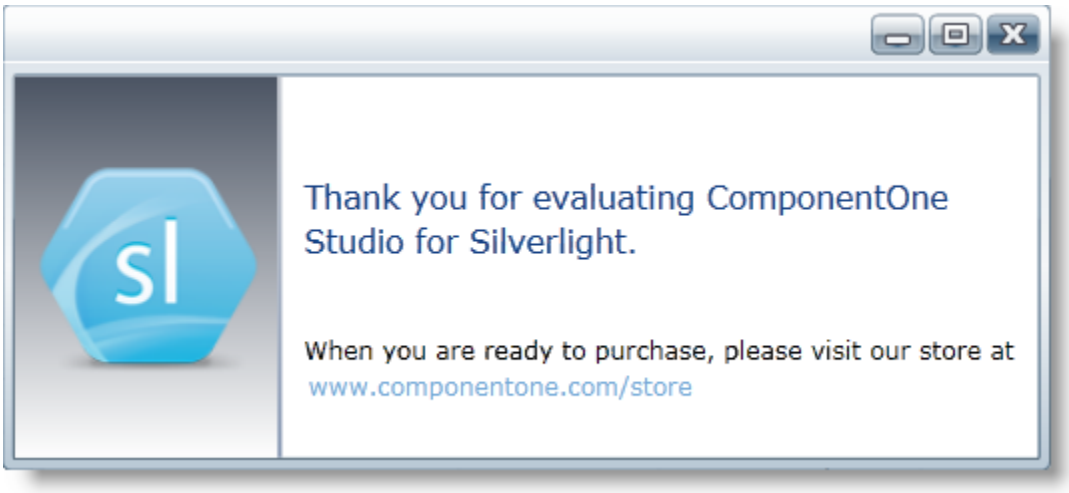
Licensing for **ComponentOne Studio for Silverlight** is similar to licensing in other ComponentOne products but there are a few differences to note.

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 Studio for Silverlight**, when a control is dropped on a form, a license nag dialog box appears one time.

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 e-mail will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Product Forums**

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

- **Installation Issues**

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

- **Documentation**

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

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

Redistributable Files

ComponentOne Chart3D 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.Chart.dll
- C1.Silverlight.Chart.Editor.dll

- C1.Silverlight.Chart.Extended.dll
- C1.Silverlight.Chart3D.dll

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

About This Documentation

You can create your applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

Acknowledgements

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.

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.

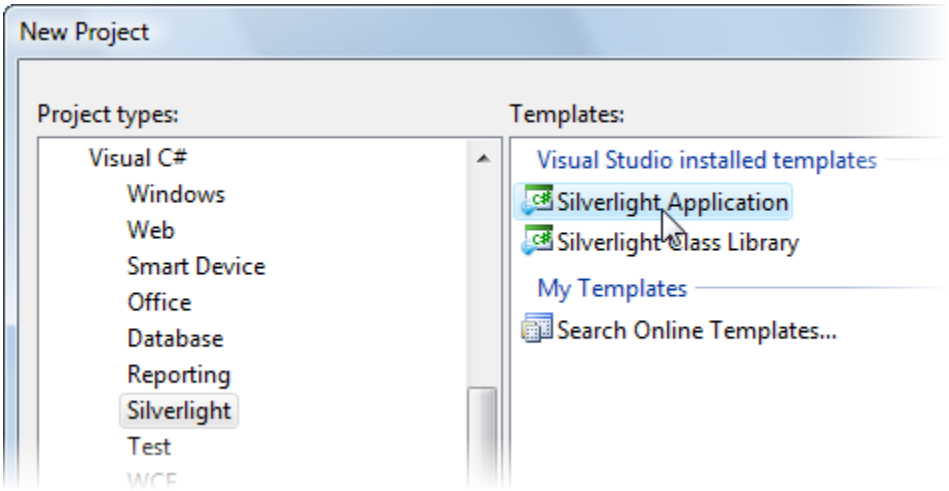
Creating a New Silverlight Project in Visual Studio

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

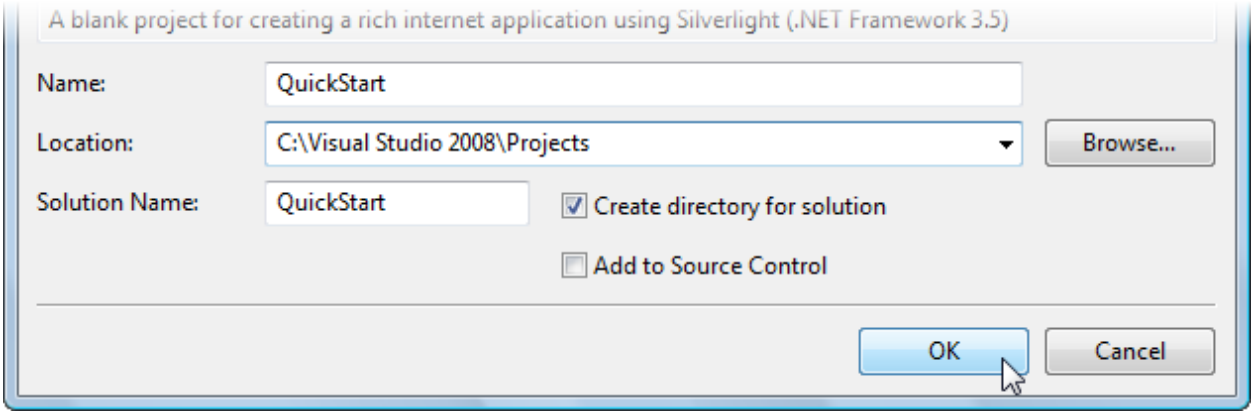
In Visual Studio 2008

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.

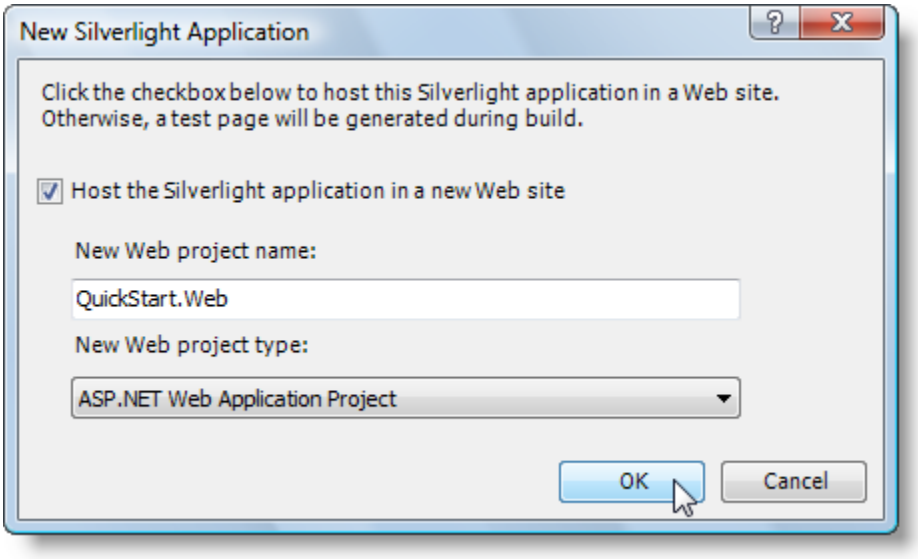


4. Name the project, specify a location for the project, and click **OK**.



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

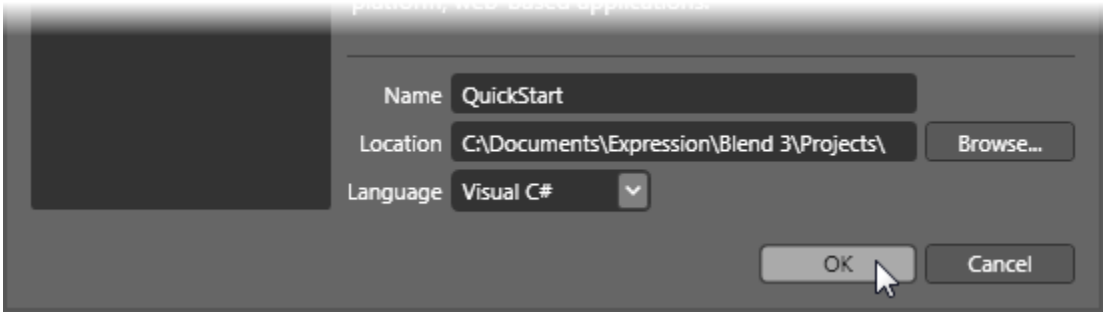
5. In the **NewSilverlight Application** dialog box, select **OK** to accept the default name and options and to create the project.



Creating a New Silverlight Project in Expression Blend

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.

Adding the Chart3D for Silverlight Components to a Blend Project

To add a reference to the assembly:

1. Select **Project | Add Reference**.
1. Browse to find the **C1.Silverlight.Chart3D.dll** assembly installed with **Chart3D for Silverlight**.
2. Select **C1.Silverlight.Chart3D.dll** and click **Open**. A reference is added to your project.

To add a component from the Asset Library:

1. Once you have added a reference to the **C1.Silverlight.Chart3D.dll**, click the **Asset Library** button  in the Blend Toolbox. The **Asset Library** appears.
2. Click the **Controls** drop-down arrow and select **All**.
3. Select **C1Chart3d**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1Chart3D** component in the Toolbox to add it to **Window1.xaml**.

Adding the Chart3D for Silverlight Components to a Visual Studio Project

When you install **ComponentOne Chart3D for Silverlight** the C1Chart3D control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne Chart3D for Silverlight provides the following control:

- C1Chart3D

To use a **Chart3D for Silverlight** control, add it to the window or add a reference to the **C1.Silverlight.Chart3D** assembly to your project.

Manually Adding Chart3D for Silverlight to the Toolbox

When you install **Chart3D for Silverlight**, the following **Chart3D for Silverlight** control will appear in the Visual Studio Toolbox customization dialog box:

- C1Chart3D

To manually add the C1Chart3D control to the Visual Studio Toolbox, complete the following steps:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu, if necessary) and right-click the Toolbox to open its context menu.
2. To make **Chart3D for Silverlight** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1Chart3D**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu.

   The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **Silverlight Components** tab.
5. Sort the list by Namespace (click the *Namespace* column header) and select the check boxes for components belonging to the **C1.Silverlight** namespace. Note that there may be more than one component for each namespace.

Adding Chart3D to the Window

To add **ComponentOne Chart3D for Silverlight** to a window or page, complete the following steps:

1. Add the C1Chart3D control to the Visual Studio Toolbox.
2. Double-click C1Chart3D or drag the control onto the window.

Adding a Reference to the Assembly

To add a reference to the **Chart3D for Silverlight** assembly, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne Chart3D for Silverlight** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.Silverlight.Chart3D.dll** assembly and click **OK**.
3. Double-click the window caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

```
Imports C1.Silverlight.Chart3D
```

   This makes the objects defined in the **Chart3D for Silverlight** assembly visible to the project.

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 **C1ComboBoxe**s. 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).

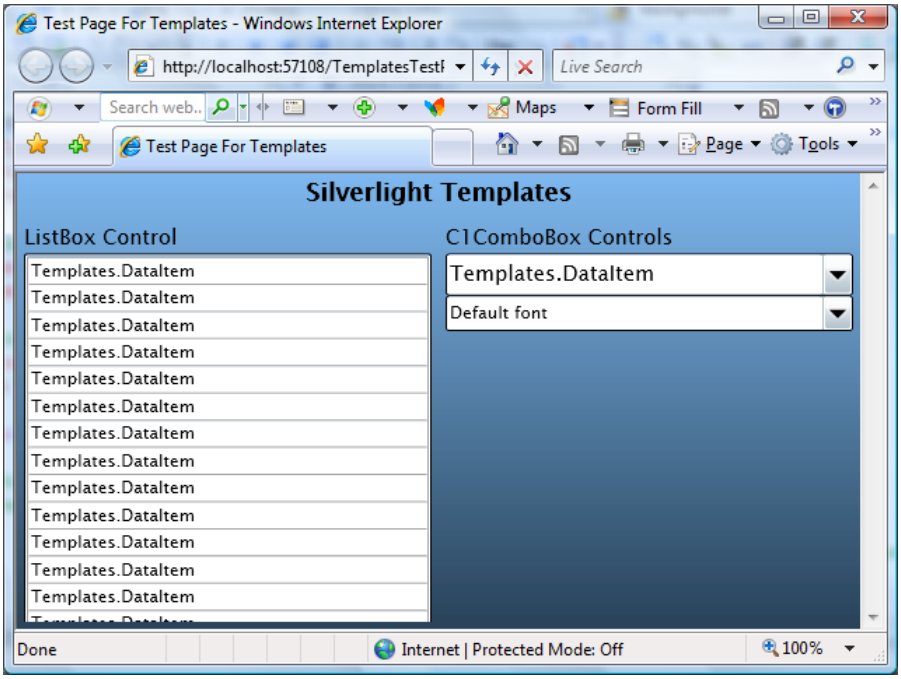
2. Add the following code to implement the **GetItems()** method in the MainPage class:

```
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.

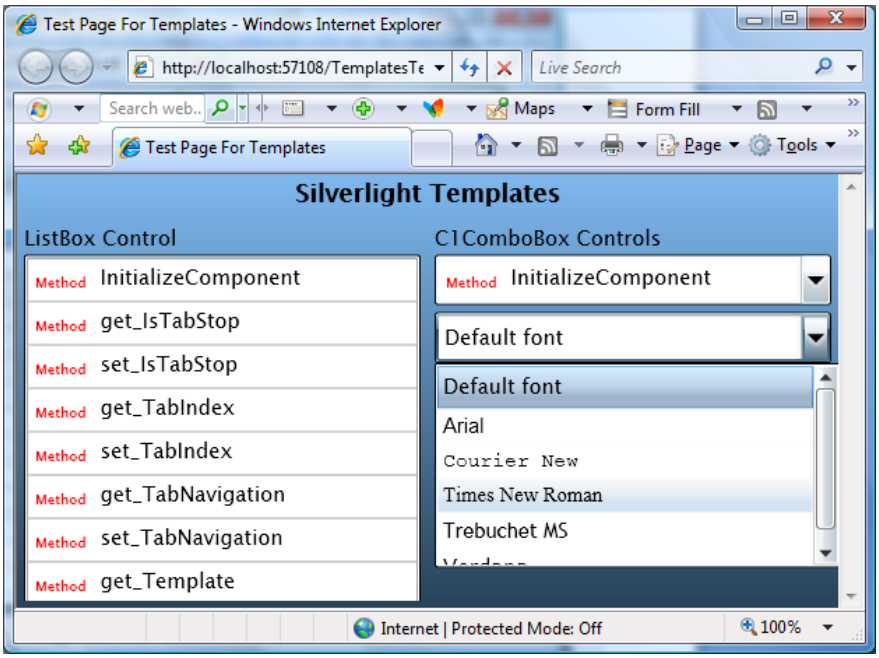
This time, we will not define the template as a resource. It will only be used in one place, so we can insert it inline, as shown below:

```
<!-- 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:

Default

Bubbly

Red

Flat

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.

Key Features

Chart3D for Silverlight provides the following unique features:

- **Surface Chart Types**

  Chart3D for Silverlight includes 6 different surface chart types. You can choose among surface charts with wire frames, contour levels, and zones when you create your chart. Customize whether contours and meshes appear along each axis. See Chart Types (page 25) for more information.

- **Show a Legend**

  Display a chart legend for zoned charts. See Adding a Chart Legend (page 25) for more information.

- **Set Rotation and Elevation**

  Rotate the chart to any angle by setting the Azimuth and Elevation properties. See Rotating and Elevating a Chart (page 26) for more information.

- **Floors and Ceilings**

  The contours and zones determined for the chart can be displayed on the ceiling or floor of the plot cube. Display contours and zones on the top and bottom of the chart to give a 2D representation in addition to the 3D model. See Adding Floors and Ceilings (page 27) for examples.

- **Show 2D Projection**

  Easily create flat (or two-dimensional) 3D charts such as heat maps. See Creating a Two-Dimensional Chart (page 29) for more information.

- **Create Custom Axis Annotations**

  Customize the exact layout and values of the axis labels using the Axis3D AnnoTemplate property.

Chart3D for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Chart3D for Silverlight**. This quick start will guide you through the basic steps for creating a typical chart: adding a 3D chart to your project, adding data to the chart, selecting a chart type and changing the chart's appearance, and adding a legend for the chart.

Note that this quick start provides the steps used in Visual Studio 2010. The XAML can be used in Microsoft Blend as well, although the steps may be slightly different.

Step 1 of 5: Adding Chart3D for Silverlight to your Project

In this step you'll either begin in Visual Studio or Blend to create a chart application using **Chart3D for Silverlight**.

To add Chart for Silverlight to your Visual Studio Project:

4. Create a new Silverlight project in Visual Studio.
5. Double-click the C1Chart3D control in the Toolbox to add it to your window. The XMAL markup will look similar to the following:

```
<UserControl xmlns:c1chart3d="clr-
namespace:C1.Silverlight.Chart3D;assembly=C1.Silverlight.Chart3D"
x:Class="MySilverlightApplication.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:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
<c1chart3d:C1Chart3D />
    </Grid>
</UserControl>
```

In the next step you will add your own data to the chart.

Step 2 of 5: Adding Data

Chart3D for Silverlight supports data values defined as a two-dimensional array of z-values where the first index corresponds to X and the second index corresponds to Y. One of the most common uses of a **Chart3D** is plotting 3D functions. Let's plot a function defined as the following:

z(x,y) = x*x - y*y;

in the range

-1 <= x <= 1

-1 <= y <= 1

To do this, follow these steps:

6. Select **View | Code** in your project.
7. Add the following statement at the top of the page:

```
using C1.WPF.C1Chart3D;
```

8. Add the following code to define the data:

```
public MainWindow()
        {
            InitializeComponent();
            c1Chart3D1.Children.Clear();

            // create 2D array 10x10
            int xlen = 10, ylen = 10;
            var zdata = new double[xlen, ylen];
            double stepx = 2.0 / (xlen - 1);
            double stepy = 2.0 / (ylen - 1);

            // calculate function for all points in the range
            for (int ix = 0; ix < xlen; ix++)
                for (int iy = 0; iy < ylen; iy++)
                {
                    double x = -1.0 + ix * stepx; //  -1 <= x <= 1
                    double y = -1.0 + iy * stepy; //  -1 <= x <= 1
                    zdata[ix, iy] = x * x - y * y;
                }

            // create data series
            var ds = new GridDataSeries();
            ds.Start = new Point(-1, -1); // start for x,y
            ds.Step = new Point(stepx, stepy); // step for x,y
            ds.ZData = zdata; // z-values

            // add series to the chart
            c1Chart3D1.Children.Add(ds);
        }
```

In the next step you will change the appearance of the chart.

Step 3 of 5: Changing the Chart Appearance

In this step you will change the chart type to **SurfaceZone**, set the appearance for the chart floor, rotate the chart, and change its elevation angle.

9. Select the C1Chart3D control in your window and click **View | Properties Window**.
10. Change the chart type by clicking the drop-down arrow next to the ChartType property and selecting **SurfaceZone**.
11. Click the drop-down arrow next to the FloorAppearance property and select **ZoneContour**.
12. Enter **170** next to the Elevation property.
13. Set the Azimuth property to **20**.

The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1chart3d="clr-
namespace:C1.Silverlight.Chart3D;assembly=C1.Silverlight.Chart3D"
x:Class="SilverlightApplication12.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:DesignHeight="300" d:DesignWidth="400">
```

```
    <Grid x:Name="LayoutRoot" Background="White">
<c1chart3d:C1Chart3D ChartType="SurfaceZone"
FloorAppearance="ZoneContour" Azimuth="20" Elevation="170">
            <c1chart3d:C1Chart3D />
    </Grid>
</UserControl>
```

In the next step you will add a legend for the chart.

Step 4 of 5: Adding a Legend

In this step you will add a legend for the chart using only one property.

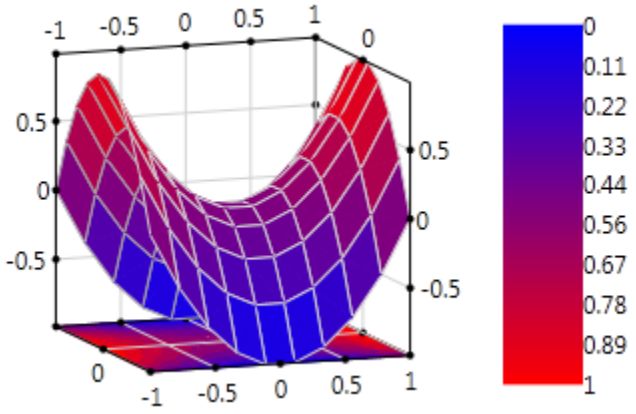
14. Select the C1Chart3D control in your window and click **View | Properties Window**.
15. Add a legend by clicking the drop-down arrow next to the Legend property and selecting **C1Chart3DLegend**.

In the next step you will run the project and view your new chart.

Step 5 of 5: Running the Project

In this step, you will run the project to see the chart.

Select **Start | Debugging** or press the **F5** key. You will see the **Chart3D** and legend.



Congratulations! You've completed the **Chart3D for Silverlight** quick start.

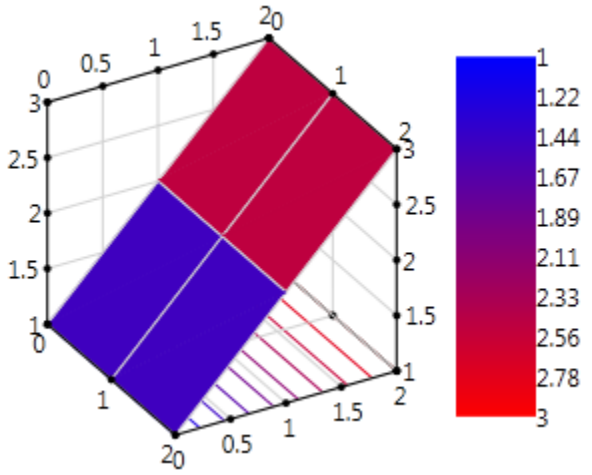
XAML Quick Reference

This section provides an example that shows how to use the Silverlight **C1Chart3D** control with only XAML code.

EX: Set up a 3D Surface Chart

The following XAML shows how to declare the **C1Chart3D** control, set the **ChartType**, **FloorAppearance**, **GridDataSeries**, and add a **C1Chart3DLegend** control. The XAML can be added inside the <Grid>…</Grid> tags.

The following chart is produced using the following XAML code:




```
<Grid>
    <c1chart3d:C1Chart3D Name="c1Chart3D1" ChartType="SurfaceZone"
FloorAppearance="Contour">
        <c1chart3d:C1Chart3D.Legend>
            <c1chart3d:C1Chart3DLegend />
        </c1chart3d:C1Chart3D.Legend>
        <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3" />
    </c1chart3d:C1Chart3D>
</Grid>
```

Using Chart3D for Silverlight

Graph your data in three dimensions. With **ComponentOne Chart3D for Silverlight** you can create professional looking 3D surface charts with options for contour levels and zones. Rotate the chart to any angle, show a chart legend and more.

Data Layout in the GridDataSeries

Data in **Chart3D for Silverlight** is defined in the GridDataSeries class. It provides the following properties related to data:

Property	Description
ZData	Gets or sets two-dimensional array of values on the grid.
Start	Gets or sets the start point of data.
Step	Gets or sets the step of grid data.
ContourData	Gets or sets two-dimensional array of contour data (4-dimensional chart).

The first index of the 2D array in the ZData/ ContourData properties corresponds to the X and the second index corresponds to the Y. The layout for ZData is shown in the following table.

	Start.X	Start.X+Step.X	Start.X+2*Step.X
Start.Y	ZData[0,0]	ZData[1,0]	ZData[2,0]
Start.Y+Step.Y	ZData[0,1]	ZData[1,1]	ZData[2,1]
Start.Y+2*Step.Y	ZData[0,2]	ZData[1,2]	ZData[2,2]

The element of the array ZData[0,0] corresponds to the point (Start.X, Start.Y), ZData[1,0] is the z-value at (Start.X+Step.X, Start.Y), and so on.

Chart Types

Using built-in types is the simplest way to set up the chart's appearance. For example, to set up a wire-frame surface chart, specify the corresponding string in the ChartType property:

```
<c1chart3d:C1Chart3D ChartType="SurfaceWireframe"/>
```

The available chart types are specified by the members of enumeration Chart3DType.

The list of available built-in chart types is presented in the table below.

Name	Description
SurfaceWireframe	Wire-frame surface chart
SurfaceWireframeContour	Wire-frame surface chart with contour levels
Surface	Surface chart
SurfaceCountour	Surface chart with contour levels
SurfaceZone	Surface chart with zones
SurfaceZoneContour	Surface chart with contour zones

Surface charts examine the distribution of data and draw contour lines to demarcate each of the contour levels. You can set the number of contour levels in the chart using the ContourLevelsCount property.

Chart3D for Silverlight also supports mesh lines which can be added to the X-axis, Y-axis, or both axes. Use the SurfaceMeshAppearance property to determine where the mesh lines appear.

Adding a Chart Legend

Chart3D for Silverlight allows you to show a legend for zoned charts. The legend element displays information about each data series of the chart. The chart legend displays the mapping between the physical colors and the data series. It is controlled by the Legend property of C1Chart3D.

To add a legend for the chart, follow these steps:

16. Add a C1Chart3D control to the window.

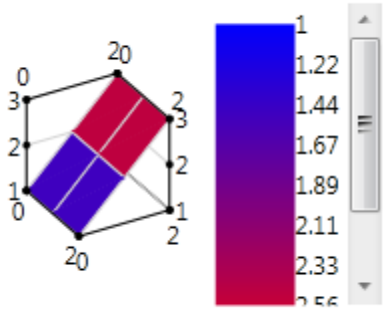
17. Set the ChartType to **SurfaceZone** or **SurfaceZoneContour**. The XAML will look similar to the following:

```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="137,77,0,0" Name="c1Chart3D1" VerticalAlignment="Top" Width="200"
ChartType="SurfaceZone">
            <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3" />
        </c1chart3d:C1Chart3D>
```

18. Set the Legend property to **C1Chart3DLegend**. Now the XAML will look like this:

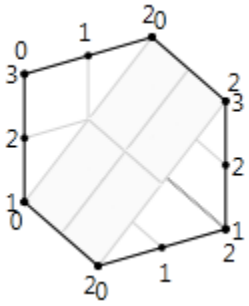
```
  <c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="178,85,0,0" Name="c1Chart3D1" VerticalAlignment="Top" Width="200"
ChartType="SurfaceZone">
            <c1chart3d:C1Chart3D.Legend>
                <c1chart3d:C1Chart3DLegend />
            </c1chart3d:C1Chart3D.Legend>
            <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3" />
        </c1chart3d:C1Chart3D>
```

Your window will look similar to the following:



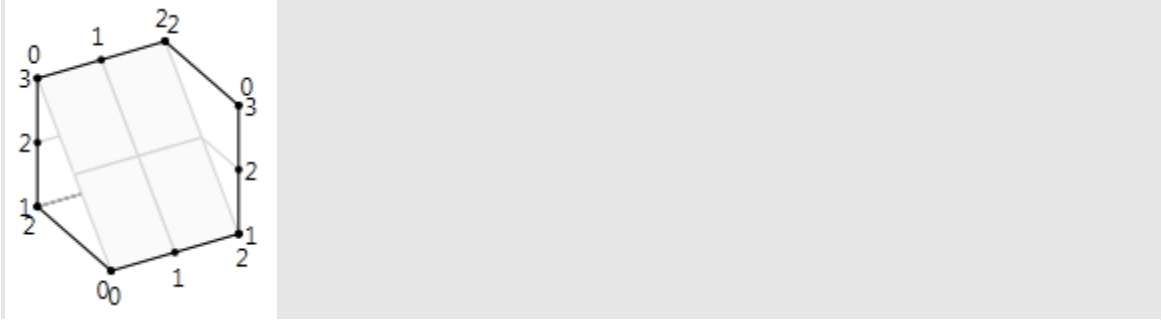
Rotating and Elevating a Chart

It's easy to rotate a chart using the Azimuth property. The default angle for C1Chart3D is 30 degrees.

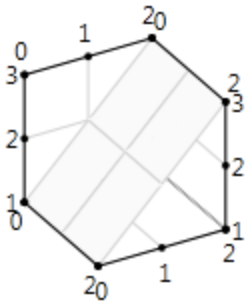


The following XAML sets the Azimuth property to 120 degrees.

```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="96,51,0,0" Name="c1Chart3D1" VerticalAlignment="Top"
Width="200" Azimuth="120">
```

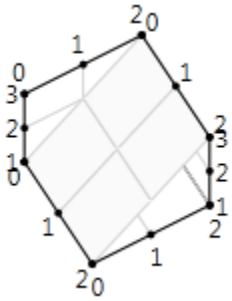


Each chart has an elevation angle, or an incline above the X-axis. When you first add a chart to the window, the elevation angle is 150 degrees, by default.



To change the incline angle, use the Elevation property. The following XAML sets the Elevation property to 120.

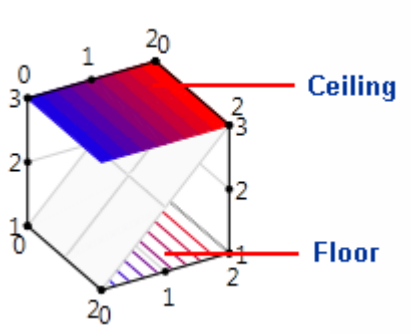
```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="96,51,0,0" Name="c1Chart3D1" VerticalAlignment="Top"
Width="200" Azimuth="30" Elevation="120">
```



You can also create a two-dimensional chart using the Elevation property. See Creating a Two-Dimensional Chart (page 29) for more information.

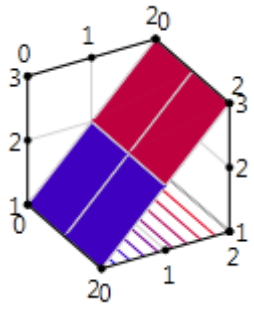
Adding Floors and Ceilings

Chart3D for Silverlight allows you to give your charts a 2D representation by displaying contours and zones on the top, or ceiling, and bottom, or floor, of the plot cube of the chart.



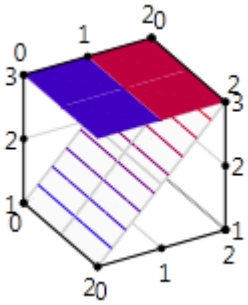
To determine the appearance of the floor, set the FloorAppearance property. In the following example, the XAML sets the FloorAppearance of a **SurfaceZone** chart to **Contour**.

```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="234,132,0,0" Name="c1Chart3D1" VerticalAlignment="Top"
Width="200" ChartType="SurfaceZone" FloorAppearance="Contour">
          <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3" />
</c1chart3d:C1Chart3D>
```



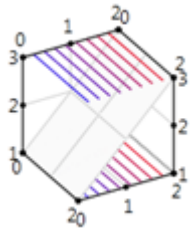
To determine the appearance of the ceiling, set the CeilAppearance property. In the following example, the XAML sets the CeilAppearance of a **SurfaceContour** chart to **Zone**.

```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="234,132,0,0" Name="c1Chart3D1" VerticalAlignment="Top"
Width="200" ChartType="SurfaceContour" CeilAppearance="Zone">
            <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3"
/>
</c1chart3d:C1Chart3D>
```

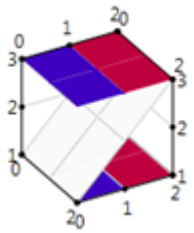


The FloorAppearance and CeilAppearance properties offer four appearance options:

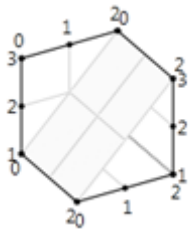
Contour **Zone**

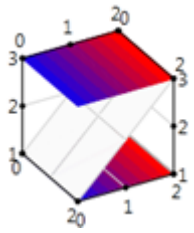


None



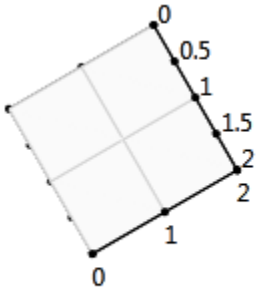
ZoneContour



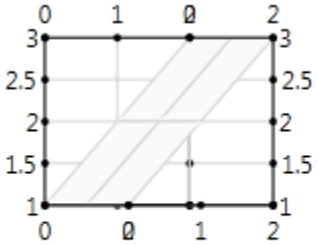


Creating a Two-Dimensional Chart

You can easily create a flat 3D chart, such as a heat map, by changing the Elevation property of a chart. Each chart has an elevation angle, or an incline above the X-axis. When you first add a chart to the window, the elevation angle is 150 degrees, by default. You can flatten the chart by setting the Elevation property to **90** or **180**, depending on how you want to flatten it with respect to the X-axis.



Elevation = "90"



Elevation = "180"

The following XAML sets the Elevation property for a chart to **90**.

```
<c1chart3d:C1Chart3D Height="150" HorizontalAlignment="Left"
Margin="234,139,0,0" Name="c1Chart3D1" VerticalAlignment="Top"
Width="200" Elevation="90">
            <c1chart3d:GridDataSeries ZDataString="1 1 1,2 2 2,3 3 3"
/>
        </c1chart3d:C1Chart3D>
```

Creating Custom Axis Annotations

The 3D chart has an X, Y, and Z axis that can be controlled by the C1Chart3D object. You can customize the look and feel of the axes through the C1Chart3D object and then set the X, Y, or Z axis object through the AxisX, AxisY, and AxisZ properties for the new Axes settings to appear on the X, Y, and Z axes.

This section describes the most common axis configuration scenarios used to make the chart more readable such as labeling, scaling, and formatting.

Custom Axis Annotation Template

The following sample shows how to use the Axis3D.AnnoTemplate property to create axis labels with color depending on the corresponding axis value.

Add the following XAML to your MainWindow.xaml:

```
<UserControl xmlns:c1chart3d="clr-
namespace:C1.Silverlight.Chart3D;assembly=C1.Silverlight.Chart3D"
x:Class="MySilverlightApplication.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:DesignHeight="300" d:DesignWidth="400">
  <UserControl.Resources>
    <!-- instance of converter -->
    <local:LabelToColorConvereter x:Key="cnv" />
  </UserControl.Resources>
    <Grid>
    <c1chart3d:C1Chart3D Name="c1Chart3D1" >
      <c1chart3d:GridDataSeries ZDataString="-1 -1 -1,0 0 0, 1 1 1" />
      <c1chart3d:C1Chart3D.AxisZ>
        <c1chart3d:Axis3D>
          <!-- set axis annotation template  -->
          <c1chart3d:Axis3D.AnnoTemplate>
            <DataTemplate >
              <!-- DataContext is axis label(string), use converter to set color
-->
                <TextBlock Width="20" Height="12" Text="{Binding}"
Foreground="{Binding Converter={StaticResource cnv}}" />
            </DataTemplate>
          </c1chart3d:Axis3D.AnnoTemplate>
        </c1chart3d:Axis3D>
      </c1chart3d:C1Chart3D.AxisZ>
    </c1chart3d:C1Chart3D>
  </Grid>
</UserControl>
```

Select **View | Code** and add the following code. The binding converter selects the brush of the axis label depending on the value.

```
public class LabelToColorConvereter : IValueConverter
{
  static Brush belowZero = new SolidColorBrush(Colors.Blue);
  static Brush aboveZero = new SolidColorBrush(Colors.Red);
  static Brush zero = new SolidColorBrush(Colors.DarkGray);

  public object Convert(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
  {
    // string representing axis label
    string s = value as string;
    if (!string.IsNullOrEmpty(s))
    {
      var dv = double.Parse(s);

      // return brush depending on the value
      if (dv < 0)
        return belowZero;
      else if (dv > 0)
        return aboveZero;
      else
        return zero;
    }

    return value;
  }

  public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
  {
    throw new NotImplementedException();
  }
}
```

Axis Title

Adding a title to an axis clarifies what is charted along that axis. A title with a specified font can be added to any axis.

To Add an Axis Title:

Use the axis Title property to add a title to an axis.

19. In the Visual Studio Properties window, expand the AxisX, AxisY, or AxisZ property.
20. Enter a title next to the **Title** property.

To remove the title, delete the text.