
ComponentOne

Toolbar 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 ToolBar for Silverlight Overview	1
Installing Toolbar for Silverlight.....	1
Toolbar for Silverlight Setup Files.....	1
System Requirements.....	2
Installing Demonstration Versions.....	2
Uninstalling Toolbar for Silverlight	2
Using Maps Powered by Esri.....	2
End-User License Agreement	3
Licensing FAQs	3
What is Licensing?	3
Studio for Silverlight Licensing	4
Technical Support.....	5
Redistributable Files	5
About This Documentation	6
Silverlight Resources	6
Creating a Silverlight Project in Visual Studio	7
Creating a Silverlight Project in Expression Blend.....	8
Adding the Toolbar for Silverlight Components to a Blend Project	8
Adding the Toolbar for Silverlight Components to a Visual Studio Project	8
Using Templates	10
Data Templates.....	10
Control Templates.....	15
Preparing Your Enterprise Environment	18
Key Features.....	19
Toolbar for Silverlight Quickstart	20
Step 1 of 4: Adding Toolbar for Silverlight to your Project	20
Step 2 of 4: Adding C1ToolBarGroups to C1ToolBar.....	21
Step 3 of 3: Adding a C1ToolBarStrip and C1ToolBarToggleButtons to C1ToolBarGroup	22
XAML Quick Reference	23
EX: Add Items to the C1ToolBar	23

Toolbar Elements	24
Toolbar Group	24
Toolbar Button	25
Toolbar DropDown	25
Toolbar SplitButton	26
Toolbar Strip	27
Toolbar Tab Item	28
Toolbar ToggleButton	30
Toolbar for Silverlight Layout and Appearance	31
Toolbar Layout	31
Button Size and Text Position in C1ToolbarGroup	32
Toolbar Themes	32
Toolbar for Silverlight Appearance Properties	33
C1Toolbar Templates	34
C1Toolbar Styles	35
Toolbar ClearStyle Properties	35
Toolbar for Silverlight Samples	35
Toolbar for Silverlight Task-Based Help	36
Aligning Toolbar Buttons	36
Adding an Image to the Toolbar Button	36
Adding Logic Behind the ToolbarButton Click Event	36
Commanding with C1Toolbar for Silverlight	39
Using C1ToolbarCommand	39

ComponentOne ToolBar for Silverlight Overview

Create custom toolbars to provide additional navigation in your website with **ComponentOne Toolbar™ for Silverlight**. **Toolbar for Silverlight** includes one control, **C1Toolbar**, which includes items (such as links, custom content, and separators) and groups, giving you the flexibility to place almost any control in the toolbar.

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)

- [Toolbar for Silverlight Layout and Appearance](#) (page 31)

- [C1Toolbar Templates](#) (page 34)

Installing Toolbar for Silverlight

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

Toolbar for Silverlight Setup Files

The **ComponentOne Studio 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\\My Documents\ComponentOne Samples\Studio for Silverlight

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

See the [Toolbar for Silverlight Samples](#) (page 35) 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](#) or visit the Esri website at <http://www.esri.com> for additional information.

System Requirements

System requirements for **ComponentOne Toolbar 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 Studio 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 Toolbar for Silverlight

To uninstall **ComponentOne Toolbar for Silverlight**:

Open the **Control Panel** and select **Add or Remove Programs (XP)** or **Programs and Features (Windows 7/Vista)**.

Select **ComponentOne Toolbar for Silverlight 4.0** and click the **Remove** button.

Click **Yes** to remove the program.

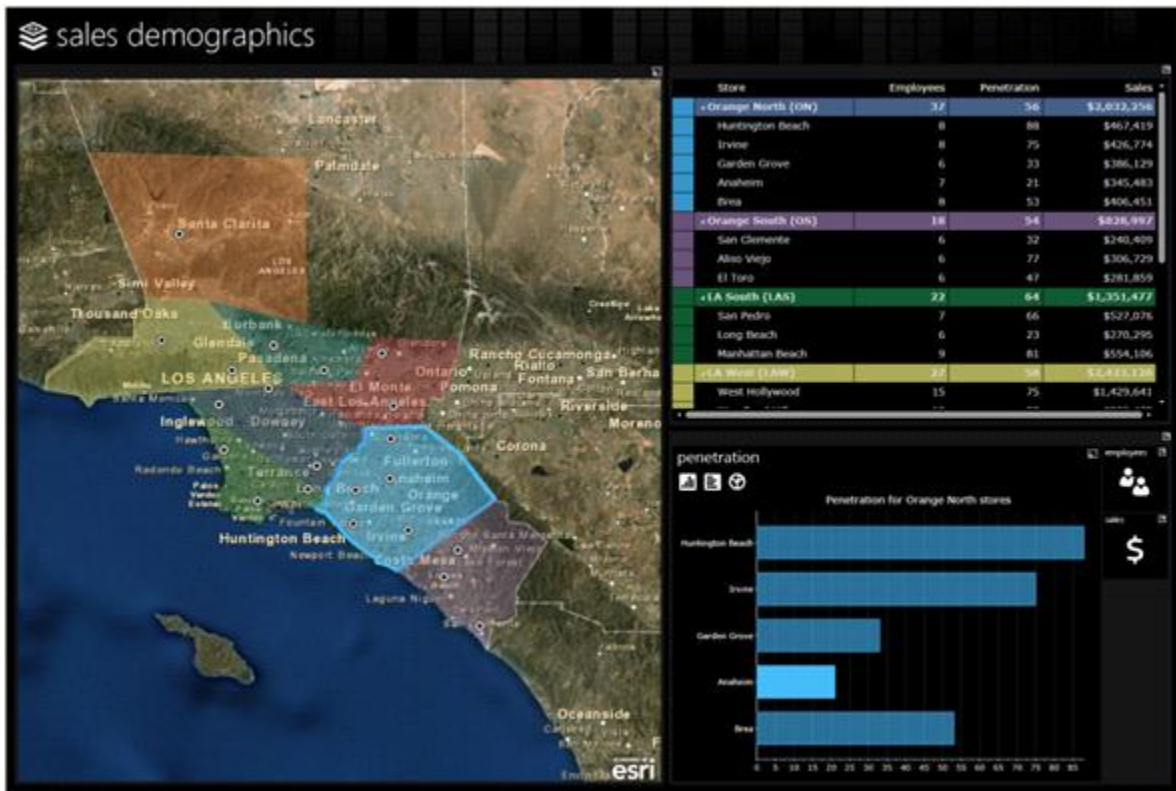
Using Maps Powered by Esri

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 Windows Phone Setup Files](#) topic for more information on the Esri files installed with this product.

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

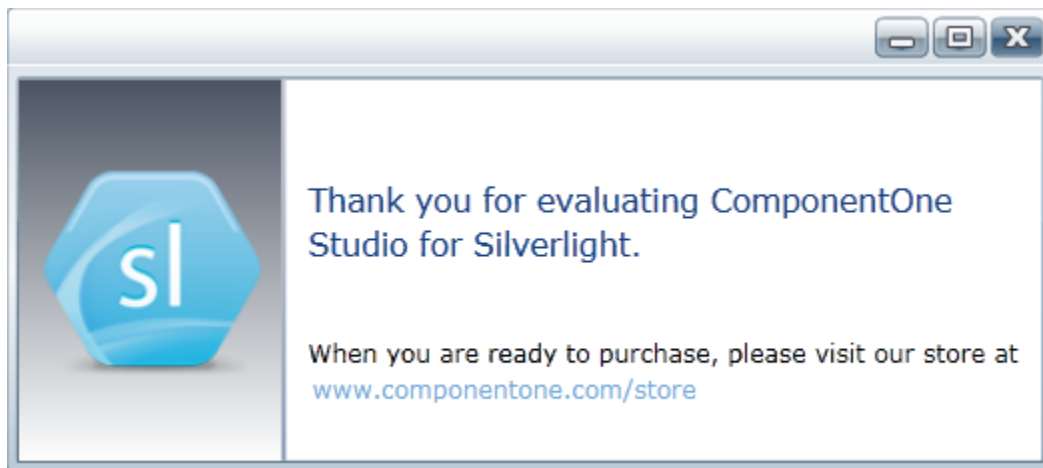
Licensing for **ComponentOne Studio for Silverlight** is similar to licensing in other ComponentOne products but there are a few differences to note.

Initially licensing is 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 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 Toolbar 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.Toolbar.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

Microsoft, Windows, Windows Vista, Visual Studio, and Microsoft Expression are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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/corinab/archive/2008/03/11/silverlight-2-control-skins.aspx>
A practical discussion of skinning Silverlight controls and applications by Corrina Barber.

Creating a Silverlight Project in Visual Studio

Complete the following steps to create a new Silverlight project in Microsoft Visual Studio 2010:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2010.
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 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 Toolbar 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.Toolbar.dll** assembly installed with **Toolbar for Silverlight**.

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

2. Select **C1.Silverlight.Toolbar.dll** and click **Open**. A reference is added to your project.

To add a component from the Asset Library:

1. Once you have added reference to the **C1.Silverlight.Toolbar.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 **C1Toolbar**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1Toolbar** component in the Toolbox to add it to **Window1.xaml**.

Adding the Toolbar for Silverlight Components to a Visual Studio Project

When you install **ComponentOne Toolbar for Silverlight** the **C1Toolbar** control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne Toolbar for Silverlight provides the following controls:

- **C1Toolbar**
- **C1ToolbarButton**
- **C1ToolbarDropDown**

- C1ToolBarGroup
- C1ToolBarStrip
- C1ToolBarTabItem
- C1ToolBarTabControl
- C1ToolBarToggleButton

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

Manually Adding Toolbar for Silverlight to the Toolbox

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

- C1ToolBar
- C1ToolBarButton
- C1ToolBarDropDown
- C1ToolBarGroup
- C1ToolBarStrip
- C1ToolBarTabItem
- C1ToolBarTabControl
- C1ToolBarToggleButton

To manually add the C1ToolBar 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 **ToolBar for Silverlight** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1ToolBar**, 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 Toolbar to the Window

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

1. Add the C1ToolBar control to the Visual Studio Toolbox.
2. Double-click C1ToolBar or drag the control onto the window.

Adding a Reference to the Assembly

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

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne Toolbar for Silverlight** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.Silverlight.Toolbar.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.Toolbar`

This makes the objects defined in the **Toolbar 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 **C1ComboBoxes**. 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;
}

```

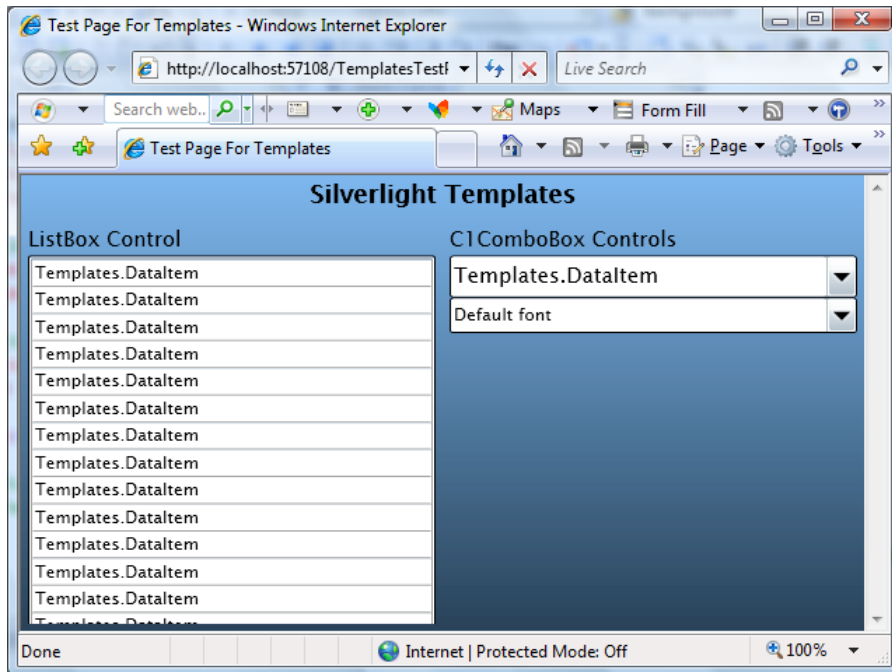
3. Add the definition of the **DataItem** class. to the **MainPage.xaml.cs** file, below the **MainPage** class definition:

```

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 **TextBlocks** 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:

```

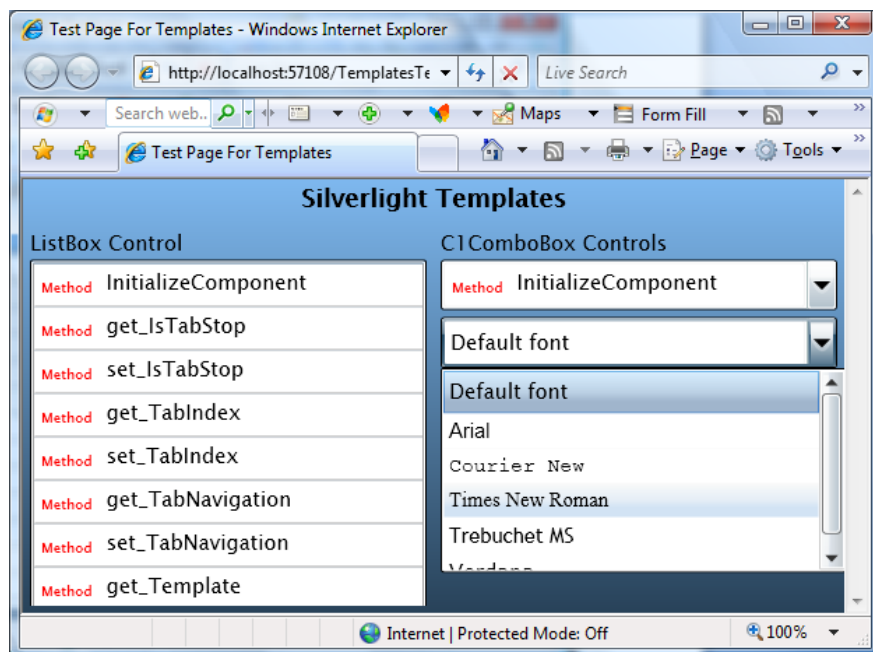
<!-- C1ComboBox 2 -->
<C1_Silverlight:C1ComboBox x:Name="_cmb2" FontSize="12" Margin="0,0,0,5" >
  <C1_Silverlight:C1ComboBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding}" FontFamily="{Binding}" Margin="4" />
    </DataTemplate>
  </C1_Silverlight:C1ComboBox.ItemTemplate>
</C1_Silverlight:C1ComboBox>

```

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

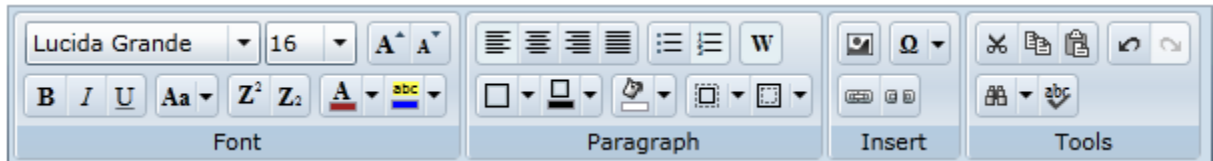
The [Silverlight Enterprise Deployment Guide](#) is available for download from the Silverlight whitepapers site: <http://silverlight.net/learn/whitepapers.aspx>.

Key Features

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

- **Ribbon-like Toolbar**

Create an advanced Microsoft Ribbon-like toolbar.



- **Lightweight ToolbarStrip**

Create a lightweight C1ToolBarStrip that can be used separately, for simple scenarios.

- **ToolBar Group**

The C1ToolBar control is a container that supports any UIElement including a C1ToolBarStrip. This is used to group similar toolbar buttons such as the “Font” group in Microsoft Word. The C1ToolBarGroup has a Header portion which is displayed as a band at the bottom with text.

For more information see [ToolBar Elements](#) (page 24).

- **Overflow Support**

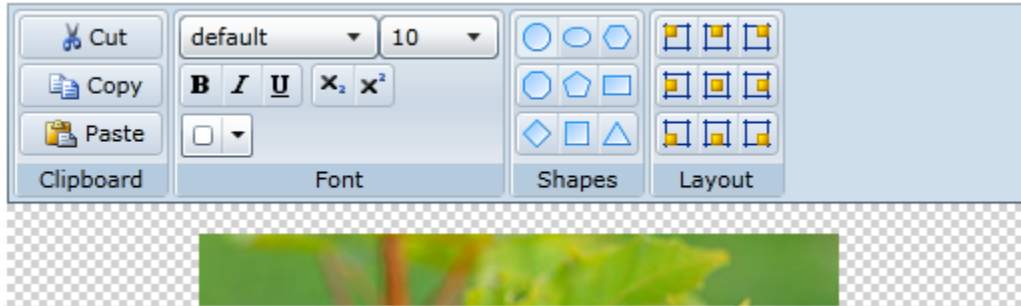
Depending on the available space in the Strip panel, the items will jump between the Strip panel and drop-down Overflow panel. This occurs automatically by default, but can be set to occur never, always, or as needed.

For more information see [ToolBar Strip](#) (page 27).



- **Personalize Your Toolbar**

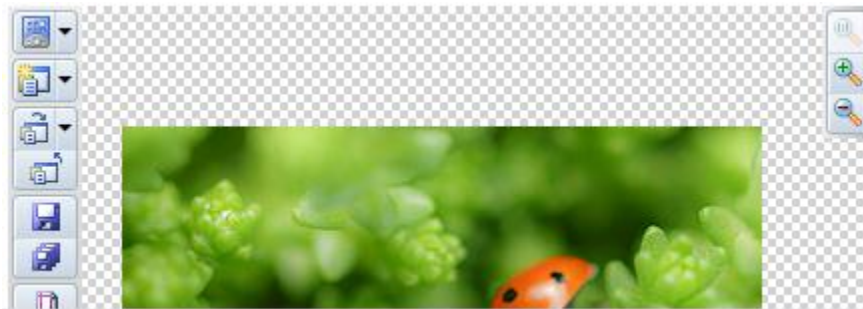
Place a toolbar button, toggle button, drop-down button, or drop-down inside a C1ToolBarStrip. Attach a behavior to each toolbar item adding simple event handlers to them.



- **Change the Orientation**

Select from horizontal (default) or vertical orientations for your toolbar.

For more information see [Toolbar Layout](#) (page 31).



- **Add Separators**

Draw a line, horizontal or vertical, between items in the toolbar. The separator helps to organize your toolbar.

- **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, RainerOrange, ShinyBlue, and BureauBlack.

- **Supports ClearStyle Technology**

Toolbar for Silverlight supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **Toolbar** elements. See [Toolbar ClearStyle Properties](#) for more information on the ComponentOne ClearStyle technology.

Toolbar for Silverlight Quickstart

The following quick start guide is intended to get you up and running with **Toolbar for Silverlight**. In this quick start you'll start in Visual Studio and create a new project, add **Toolbar for Silverlight** to your application, and add C1Toolbar items such as C1ToolbarGroup, C1ToolbarStrip, C1ToolbarButton, and C1ToolbarToggleButton to your C1Toolbar. For more information see [Toolbar Elements](#) (page 24).

Step 1 of 4: Adding Toolbar for Silverlight to your Project

To set up your project and add a **C1Toolbar** control to your application, complete the following steps

1. Create a new Silverlight project in Visual Studio.
2. Add a reference to the C1.Silverlight and the **C1.Silverlight.C1Toolbar** assemblies. In the Solution Explorer right-click on **References** and select **Add Reference**. In the **Add Reference** dialog box select the Browse tab. Browse for the C1.Silverlight.C1Toolbar.dll and the C1.Silverlight and select **OK**.
3. Define the System and the **C1.WPF.C1Toolbar** prefixes.

```
xmlns:System="clr-namespace:System;assembly=mcorlib"

        xmlns:c1toolbar="clr-
namespace:C1.Silverlight.C1Toolbar;assembly=C1.Silverlight.C1Toolbar"
```

4. Add 2 Rows to the LayoutRoot Grid and set the **Height** of the first row to **Auto**.

```
<Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
</Grid>
```

5. Drop a C1Toolbar onto the page within the first row and Remove the default properties: Height="100" HorizontalAlignment="Left" Margin="174,34,0,0". Your XAML should now look like the following:

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
    <c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
</c1:C1Toolbar>
```

Step 2 of 4: Adding C1ToolbarGroups to C1Toolbar

In this step you'll continue in Visual Studio by adding C1ToolbarGroups to your C1Toolbar. You'll then add C1ToolbarButtons to the C1ToolbarGroups.

1. Right-click on the **C1Toolbar** control and select **Properties** to open its Properties window. Navigate to the **ToolbarItems** property and click on the ellipsis button to open up the **Collection Editor: ToolbarItems**.
2. Click on the **Add** button twice to add two **C1ToolbarGroups** and click **OK**. Your XAML should now look like the following:

```
<c1:C1Toolbar Name="c1Toolbar1">
    <c1:C1ToolbarGroup />
    <c1:C1ToolbarGroup />
</c1:C1Toolbar>
```

3. In XAML, set the **Header** property of each **C1ToolbarGroup** to the following: **Clipboard** and **Font**. Your XAML should now look like the following:

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
    <c1:C1ToolbarGroup Header="Clipboard"/>
    <c1:C1ToolbarGroup Header="Font"/>
</c1:C1Toolbar>
```

4. Select the **Clipboard C1ToolbarGroup** and click on the ellipsis button next to the **Items** collection editor.
5. From the **Select item** drop-down, click **Add** three times to add three **C1ToolbarButtons** to the collection.
6. Select the first **C1ToolbarButton** and expand the **Other** node in its properties window to set the **LabelText** property to **Paste** and set the other two to **Cut** and **Copy**, respectively.

- As an optional step you can set the **LargeImageSource** and/or **SmallImageSource** properties to resources found within your project or add new images.
- Click **OK** to close the **Items Collection** editor. Your XAML should now look like this:

```
<c1:C1Toolbar Name="c1Toolbar1">
  <c1:C1ToolbarGroup Header="Clipboard">
    <c1:C1ToolbarButton LabelTitle="Paste"
LargeImageSource="/ToolbarQuickstart;component/Images/Paste.png" />
    <c1:C1ToolbarButton LabelTitle="Cut"
SmallImageSource="/ToolbarQuickstart;component/Images/Cut.png" />
    <c1:C1ToolbarButton LabelTitle="Copy"
SmallImageSource="/ToolbarQuickstart;component/Images/Copy.png" />
  </c1:C1ToolbarGroup>
  <c1:C1ToolbarGroup Header="Font"/>
</c1:C1Toolbar>
```

Step 3 of 3: Adding a C1ToolbarStrip and C1ToolbarToggleButton to C1ToolbarGroup

In this step you'll continue in Visual Studio by adding a C1ToolbarStrip to your 'Font' C1ToolbarGroup and then you will add C1ToolbarToggleButton to the C1ToolbarGroup.

- Select the **Font C1ToolbarGroup** and add a **C1ToolbarStrip** in XAML.

```
<c1:C1ToolbarGroup Header="Font">
  <c1:C1ToolbarStrip />
</c1:C1ToolbarGroup>
```

The rest of your XAML should appear like the following:

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1"
  <c1:C1ToolbarGroup Header="Clipboard">
    <c1:C1ToolbarButton LabelTitle="Paste" />
    <c1:C1ToolbarButton LabelTitle="Cut" />
    <c1:C1ToolbarButton LabelTitle="Copy" />
  </c1:C1ToolbarGroup>
  <c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip />
  </c1:C1ToolbarGroup>
</c1:C1Toolbar>
```

- Select the C1ToolbarStrip and open its **Items** collection editor
- Select the **C1ToolbarToggleButton** from the **Select item** dropdown and click **Add** three three times to add three C1ToolbarToggleButton; set each **LabelTitle** property to: Bold, Italic, and Underline. Click **OK** to close the **Items** collection editor.

```
<c1:C1ToolbarGroup Header="Font">
  <c1:C1ToolbarStrip>
    <c1:C1ToolbarToggleButton LabelTitle="Bold" />
    <c1:C1ToolbarToggleButton LabelTitle="Italic" />
    <c1:C1ToolbarToggleButton LabelTitle="Underline" />
  </c1:C1ToolbarStrip>
</c1:C1ToolbarGroup>
```

Congratulations! You've now completed creating a toolbar UI using **Toolbar for WPF**.



Further topics:

- [Adding Logic Behind the ToolbarButton Click Event](#) (page 36) – This topic shows you how to use button click events to add logic behind buttons
- [Commanding with C1Toolbar for Silverlight](#) (page 39) – This tutorial demonstrates how to use C1Toolbar with commands in a Silverlight application
- [Button Size and Text Position in C1ToolbarGroup](#) (page 32) – This topic shows you how to use GroupSizeDefinitions of the C1ToolbarGroup.
- [Toolbar Tab Item](#) (page 28) – This topic shows you how to add the C1ToolbarTabItem control.

XAML Quick Reference

This section provides an example that show how to use the Silverlight Toolbar control with only XAML code.

EX: Add Items to the C1Toolbar

The following XAML code shows you how to add a C1ToolbarGroup, C1ToolbarStrip, and C1ToolbarButton to the C1Toolbar control:

```
<c1:C1Toolbar Name="c1Toolbar1">
  <c1:C1ToolbarTabControl>
    <c1:C1ToolbarTabItem Header="Home">
      <c1:C1ToolbarGroup Header="Clipboard">
        <c1:C1ToolbarButton LabelTitle="Paste"
LargeImageSource="/Images/Paste.png" />
        <c1:C1ToolbarButton LabelTitle="Cut"
SmallImageSource="/Images/Cut.png" />
        <c1:C1ToolbarButton LabelTitle="Copy"
SmallImageSource="/Images/Copy.png" />
      </c1:C1ToolbarGroup>
      <c1:C1ToolbarGroup Header="Font">
        <c1:C1ToolbarStrip>
          <c1:C1ToolbarToggleButton LabelTitle="Bold" />
          <c1:C1ToolbarToggleButton LabelTitle="Italic" />
          <c1:C1ToolbarToggleButton LabelTitle="Underline" />
        </c1:C1ToolbarStrip>
      </c1:C1ToolbarGroup>
    </c1:C1ToolbarTabItem>
  </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

The following image shows how the C1Toolbar control will appear after adding the above XAML code:



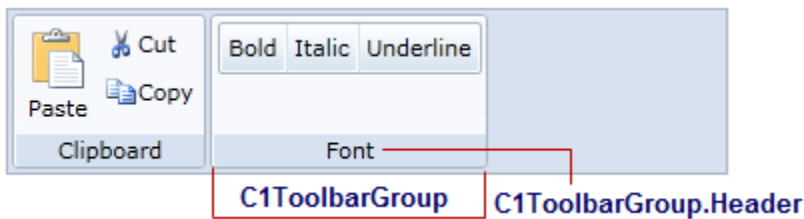
Toolbar Elements

C1ToolBar is a Silverlight control that is used as a container to hold other controls such as buttons, check buttons, text boxes, drop-down lists, split-buttons, and separators. C1ToolBar provides seven different objects to support the various types of controls used in the C1ToolBar: C1ToolBarButton, C1ToolBarDropDown, C1ToolBarGroup, C1ToolBarSplitButton, C1ToolBarStrip, C1ToolBarTabItem, C1ToolBarToggleButton,

Toolbar Group

The C1ToolBarGroup object defines a group of toolbar elements. The C1ToolBarGroup can hold the following toolbar controls: C1ToolBarButton, C1ToolBarToggleButton, C1ToolBarDropDown, and C1ToolBarSplitButton.

When you add toolbar items to the C1ToolBarGroup the child toolbar items appear grouped in a rectangular box like the following:



C1ToolBarGroups are typically used when a group of functions are mutually exclusive. That is, only one of the functions represented by the group of buttons can be on at a time.

The images of the controls (C1ToolBarButtons and C1ToolBarToggleButtons) inside the C1ToolBarGroup can be all be sized to large, medium, or small using the GroupSizeDefinitions class.

The **GroupSizeDefinition** defines the control sizes as small, medium, or large and the position of the button text in a C1ToolBarGroup. If "large" is used then all the controls (C1ToolBarButtons and C1ToolBarToggleButtons) in the group gets the image size from the LargeImageResource property and positions the text value of the LabelTitle property below the image. If "medium" is used then all the controls (C1ToolBarButtons and C1ToolBarToggleButtons) in the group gets the image size from the SmallImageResource property and positions the text value of the LabelTitle property to the right of the image. If "small" is used then all the controls (C1ToolBarButtons and C1ToolBarToggleButtons) in the group gets the image size from the SmallImageResource property and does not show the text.

EX: C1ToolBarGroup with a C1ToolBarStrip containing C1ToolBarButtons

```

</c1:C1ToolBarGroup>
<c1:C1ToolBarGroup Header="Font">
  <c1:C1ToolBarStrip>
    <c1:C1ToolBarToggleButton LabelTitle="Bold" />
    <c1:C1ToolBarToggleButton LabelTitle="Italic" />
    <c1:C1ToolBarToggleButton LabelTitle="Underline" />
  </c1:C1ToolBarStrip>
</c1:C1ToolBarGroup>

```

The C1ToolBarGroup includes the following properties:

Property	Definition
FocusBrush	Gets or sets the item that is located at a specified index of the collection.
GroupSizeDefinitions	Gets or sets the mode of selection for the items in the control.
Header	Gets or sets the header of the toolbar group.
MouseOverBrush	Gets or sets the Brush used to highlight the control when it has the mouse over.
PressedBrush	Gets or sets the Brush used to paint a button when it is pressed.
ShowDialogLauncher	Gets or sets the dialog launcher visibility.

Toolbar Button

The C1ToolBarButton object defines a toolbar button. A button can include text, image, or text and an image. Set text with the LabelTitle property, and an image with the **LargeImageSource** or **SmallImageSource** property for each C1ToolBarButton object.

The buttons can be added at design time using the **C1ToolBarItems** collection editor, programmatically using **C1ToolBarItemCollection.Add** or **C1ToolBarItemCollection.Remove** methods from the **C1ToolBarItemCollection** collection, or through XAML code..

The C1ToolBarButton includes the following unique properties:

Property	Definition
LabelTitle	Gets or sets the label title of control.
LargeImageSource	Gets or sets the large image source of the control.
MouseOverBrush	Gets or sets the Brush used to highlight the control when it has the mouse over
PressedBrush	Gets or sets the Brush used to paint a button when it is pressed.
SmallImageSource	Gets or sets the large image source of the control.

Toolbar DropDown

The C1ToolBarDropDown control represents a drop-down button on the C1ToolBarStrip. When clicking it displays popup panel with Content property or context menu set by Menu property. A drop-down button provides users with a list of options. When you click on a **C1ToolBarDropDown** button the **Click** event always fires and the drop down list appears.



Example 1: Drop-down with popup stack panel with buttons

```

<c1:C1ToolBarDropDown Padding="2" Header="Color">
  <c1:C1ToolBarDropDown.Content>
    <StackPanel Margin="2" Orientation="Horizontal">
      <Button Margin="2" Content="Red" Foreground="Red" />
      <Button Margin="2" Content="Green" Foreground="Green" />
      <Button Margin="2" Content="Blue" Foreground="Blue" />
    </StackPanel>
  </c1:C1ToolBarDropDown.Content>
</c1:C1ToolBarDropDown>

```

Example 2: Dropdown with popup menu



```

<c1:C1ToolBar Name="c1ToolBar1" Margin="0,127,0,160">
  <c1:C1ToolBarDropDown Padding="2" Header="Color">
    <c1:C1ToolBarDropDown.ContextMenu>
      <ContextMenu>
        <MenuItem Foreground="Red" Header="Red" IsCheckable="True" />
        <MenuItem Foreground="Green" Header="Green"
IsCheckable="True" />
        <MenuItem Foreground="Blue" Header="Blue" IsCheckable="True"
/>
      </ContextMenu>
    </c1:C1ToolBarDropDown.ContextMenu>
  </c1:C1ToolBarDropDown>
</c1:C1ToolBar></c1:C1ToolBarDropDown>

```

The C1ToolBarDropDown includes the following unique properties:

Property	Definition
ContentBackground	Gets or sets the content background.
Menu	Gets or sets the context menu which is displayed when the control is clicked.
MouseOverBrush	Gets or sets the Brush used to highlight the control when it has the mouse over.
PressedBrush	Gets or sets the Brush used to paint a button when it is pressed.

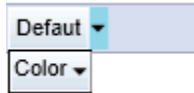
Toolbar SplitButton

C1ToolBarSplitButton control represents a drop-down split button on the C1ToolBarStrip.

It's similar to `C1ToolBarDropDown` but contains two clickable areas: the button area and the downward-pointing arrow.

When clicking on the rightmost part of the downward-pointing rectangle it displays a popup panel with the **Content** property or context menu set by the **ContextMenu** property. Clicking on the left part of button fires the **Click** event as in the standard button. Usually the **Click** event is used to perform the default or last action while the popup allows to select alternative options.

A solid vertical line dividing the image from the drop down arrow appears when you hover the cursor over the button like in the following image:



EX: Split button with popup menu

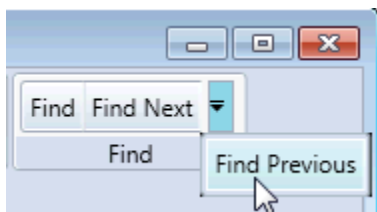
```
<c1:C1ToolBarSplitButton Padding="2" Header="Default"
Click="SetDefaultStyle">
  <c1:C1ToolBarDropDown.Menu>
    <c1:C1ContextMenu>
      <c1:C1MenuItem Header="Heading 1" FontSize="14" />
      <c1:C1MenuItem Header="Heading 2" FontSize="12" />
      <c1:C1MenuItem Header="Title" FontWeight="Bold" />
      <c1:C1MenuItem Header="Subtitle" FontWeight="SemiBold"
        FontStyle="Italic" />
      <c1:C1MenuItem Header="Quote" FontStyle="Italic" />
    </c1:C1ContextMenu>
  </c1:C1ToolBarDropDown.Menu>
</c1:C1ToolBarSplitButton>
```

Toolbar Strip

Represents a container for the toolbar controls and other C1 WPF controls. The `C1ToolBarStrip` is capable of hosting `C1ToolBarButton`, `C1ToolBarToggleButton`, `C1ToolBarDropDown`, `C1ToolBarSplitButton`, `C1Separator`, `C1ComboBox`, and `C1TextBox` controls.

The `C1ToolBarStrip` supports overflow depending on the available space in the Strip panel. The items will jump between the Strip panel and drop-down Overflow panel. This occurs automatically by default, but can be set to occur never, always, or as needed

The following image illustrates the overflow support the `C1ToolBarStrip` provides:



The `C1ToolBarStrip` can be added using XAML or programmatically.

Ex: Toolbar strip with toggle buttons

```
<c1:C1ToolBarGroup Header="Font">
  <c1:C1ToolBarStrip>
    <c1:C1ToolBarToggleButton LabelTitle="Bold" />
  </c1:C1ToolBarStrip>
</c1:C1ToolBarGroup>
```

```

<c1:C1ToolBarToggleButton LabelTitle="Italic" />
  <c1:C1ToolBarToggleButton LabelTitle="Underline" />
</c1:C1ToolBarStrip>
</c1:C1ToolBarGroup>

```

The C1ToolBarStrip includes the following unique properties:

Property	Definition
ButtonBackground	Gets or sets the Brush that will be assigned to the Background of the buttons inside the control.
ButtonForeground	Gets or sets the Brush that will be assigned to the Foreground of the buttons inside the control.
FocusBrush	Gets or sets the Brush used to highlight the focused control.
MouseOverBrush	Gets or sets the Brush used to highlight the control when it has the mouse over.
Orientation	Gets or sets the orientation of the toolbar strip.
Overflow	Gets or set the value that indicates how to handle the items which do not fit to the available space.
OverflowMenuItems	Gets the collection that contains elements of overflow menu.
OverflowPanel	Gets or sets the template for overflow panel.
PressedBrush	Gets or sets the Brush used to paint a button when it is pressed.

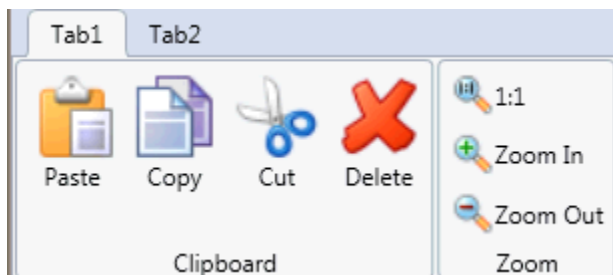
Toolbar Tab Item

The **C1ToolBarTabItem** represents a tab item on the C1ToolBar. The **C1ToolBarTabItems** are beneficial to use when you have substantial amounts of information to display on minimum window space.

The C1ToolBarTabItem divides the controls onto a separate page and shows one tab at a time.

A **C1ToolBarTabItem** can hold a collection of C1ToolBarGroups which can contain C1ToolBarButtons, C1ToolBarToggleButton, C1ToolBarDropDowns, and C1ToolBarSplitButtons.

The following image displays two **C1ToolBarTabItems**:



A **C1ToolBarTabItem** can be added to **C1ToolBar** via the **ToolBarItems** collection editor, through XAML, or programmatically.

To add a **C1ToolBarTabItem** using the designer, complete the following:

1. Add a **C1ToolBar** control to your page.
2. Select the **C1ToolBar** control and click on the ellipsis button next to the **ToolBarItems** property in the **C1ToolBar** properties window.
The **ToolBarItems** collection editor appears.
3. Select the **C1ToolBarTabItem** from the select item dropdown list and click **Add**.
The **C1ToolBarTabItem** is added to the **C1ToolBar** control.
4. Set the **Header** property to **Tab 1**.

To add a **C1ToolBarTabItem** using XAML code, add the following:

```
<c1:C1ToolBar Grid.Row="1" Name="c1ToolBar1">
  <c1:C1ToolBarTabItem Header="Tab 1">
    <c1:C1ToolBarTabItem.Content>
      <c1:C1ToolBarPanel />
    </c1:C1ToolBarTabItem.Content>
  </c1:C1ToolBarTabItem>
</c1:C1ToolBar>
```

C1ToolBarGroups can be added to the **C1ToolBarTabItem** via the **Groups** collection editor, through XAML, or programmatically using the **Groups** property.

To add a **C1ToolBarGroup** to the **C1ToolBarTabItem** using the designer, complete the following:

1. Add a **C1ToolBar** control to your page.
2. Right-click on the **C1ToolBarTabItem** and select **Properties**. Click on the **ellipsis** button next to the **Groups** property in the **C1ToolBarTabItem** properties window.
The **Groups** collection editor appears.
3. Click **Add** to add a **C1ToolBarGroup** to the **C1ToolBarTabItem**.
The **C1ToolBarGroup** is added to the **C1ToolBarTabItem**.
4. Set the **Header** property to **Tab 1**.

To add a **C1ToolBarGroup** to a **C1ToolBarTabItem** using XAML code, add the following:

```
<c1:C1ToolBar Grid.Row="1" Name="c1ToolBar1">
  <c1:C1ToolBarTabItem Header="Tab 1">
    <c1:C1ToolBarTabItem.Content>
      <c1:C1ToolBarPanel />
    </c1:C1ToolBarTabItem.Content>
    <c1:C1ToolBarGroup/>
  </c1:C1ToolBarTabItem>
</c1:C1ToolBar>
```

The **C1ToolBarTabItem** includes the following unique property:

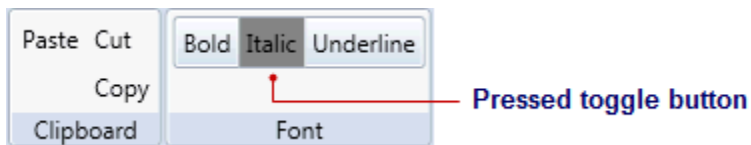
Property	Definition
Groups	Gets the collection of toolbar groups.

Toolbar ToggleButton

The `C1ToolBarToggleButton` represents a toggle button on the `C1ToolBarStrip`. It is a stateful button that enables users to toggle between on and off states. When clicking on the toggle button it remains activated, or pressed, until it is clicked again.

Toggle buttons display graphic or text like command buttons, but when it is pressed or activated the state appears in an on state. To indicate an on state you can set a color for the `PressedBrush` property.

The following image illustrates three `C1ToolBarToggleButton`s with the **Italic** toggle button pressed. The second toggle button, **Italic**, has the `PressedBrush` set to **Gray**.



`C1ToolBarToggleButton`s can be added to the `C1ToolBarStrip` via the `Items` collection editor, programmatically, or through XAML.

EX: Three `C1ToolBarToggleButton`s with one of them having an on state

```
<c1:C1ToolBar Grid.Row="1" Name="c1ToolBar1">
  <c1:C1ToolBarGroup Header="Clipboard">
    <c1:C1ToolBarButton LabelTitle="Paste" />
    <c1:C1ToolBarButton LabelTitle="Cut" />
    <c1:C1ToolBarButton LabelTitle="Copy" />
  </c1:C1ToolBarGroup>
  <c1:C1ToolBarGroup Header="Font">
    <c1:C1ToolBarStrip>
      <c1:C1ToolBarToggleButton LabelTitle="Bold" />
      <c1:C1ToolBarToggleButton PressedBrush ="Gray" LabelTitle="Italic"
    />
      <c1:C1ToolBarToggleButton LabelTitle="Underline" />
    </c1:C1ToolBarStrip>
  </c1:C1ToolBarGroup>
</c1:C1ToolBar>
```

The `C1ToolBarToggleButton` includes the following properties:

Property	Definition
GroupName	Gets or sets the name that specifies which <code>C1ToolBarToggleButton</code> controls are mutually exclusive.
LabelTitle	Gets or sets the label title of control.
LargeImageSource	Gets or sets the large image source of the control.
MouseOverBrush	Gets or sets the Brush used to highlight the control when it has the mouse over.

PressedBrush	Gets or sets the Brush used to paint a button when it is pressed.
SmallImageSource	Gets or sets the small image source of the control.

Toolbar for Silverlight Layout and Appearance

The following topics detail how to customize the C1Toolbar control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases.

You can customize the appearance of your toolbar items by using the stack panel. Inside the stack panel you can determine the orientation of your toolbar item. Additionally you can add an image and some text. The text's alignment, font style, and color can be modified.

```
<cltb:C1Toolbar>
  <cltb:C1ToolbarGroup Header="Clipboard">
    <cltb:C1ToolbarStrip Padding="0">
      <cltb:C1ToolbarButton Width="60"
HorizontalContentAlignment="Left">
        <StackPanel Orientation="Horizontal" >
          <Image Grid.Column="1" Source="Resources/cut.png" Margin="7 0
7 0"/>
          <TextBlock Text="Cut" Foreground= "DarkOrange"
VerticalAlignment="Center" TextAlignment="Center" />
        </StackPanel>
      </cltb:C1ToolbarButton>
    </cltb:C1ToolbarStrip>
    <cltb:C1ToolbarStrip Padding="0">
      <cltb:C1ToolbarButton Width="60"
HorizontalContentAlignment="Left" >
        <StackPanel Orientation="Horizontal">
          <Image Source="Resources/copy.png" Margin="4 0 4 0"/>
          <TextBlock Text="Copy" VerticalAlignment="Center"
TextAlignment="Center"/>
        </StackPanel>
      </cltb:C1ToolbarButton>
    </cltb:C1ToolbarStrip>
    <cltb:C1ToolbarStrip Padding="0">
      <cltb:C1ToolbarButton Width="60"
HorizontalContentAlignment="Left">
        <StackPanel Orientation="Horizontal">
          <Image Source="Resources/paste.png" Margin="4 0 4 0"/>
          <TextBlock Text="Paste" VerticalAlignment="Center"
TextAlignment="Center"/>
        </StackPanel>
      </cltb:C1ToolbarButton>
    </cltb:C1ToolbarStrip>
  </cltb:C1ToolbarGroup>
</cltb:C1Toolbar>
```

Toolbar Layout

Select from horizontal (default) or vertical orientations for your toolbar when you use the C1ToolbarStrip.

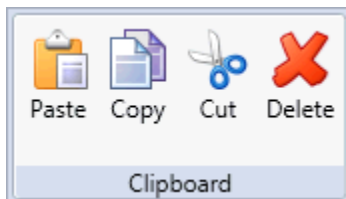
The strip panel in the C1Toolbar control can be rendered horizontally or vertically by setting its Orientation property. The layout for the individual child elements in the C1Toolbar can also be controlled using the **StackPanel.Orientation** property.

Button Size and Text Position in C1ToolbarGroup

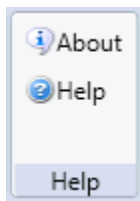
You can define the same size (small, medium, or large) for all C1ToolbarButtons and C1ToolbarToggleButtons in a C1ToolbarGroup using the **GroupSizeDefinition**.

The **GroupSizeDefinition** defines the button sizes as small, medium, or large and the position of the button text in a C1ToolbarGroup. If "large" is used then all the buttons in the group gets the image size from the LargeImageResource property and positions the text value of the LabelTitle property below the image. If "medium" is used then all buttons in the group gets the image size from the SmallImageResource property and positions the text value of the LabelTitle property to the right of the image. If "small" is used then all the buttons in the group gets the image size from the SmallImageResource property and does not show the text.

The following image displays the buttons as Large with the **LargeImageResource** property used and the text for the **LabelTitle** property shown below the image:



The following image displays the buttons as Medium with the **SmallImageResource** property used and the text for the **LabelTitle** property shown to the right of the image:



The following image displays the buttons as Small with the **SmallImageResource** property used:



Toolbar Themes

ComponentOne Toolbar for Silverlight incorporates several themes that allow you to customize the appearance of your grid. When you first add a C1Toolbar control to the page, it appears similar to the following image:

Toolbar for Silverlight Appearance Properties

ComponentOne Toolbar 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 C1Toolbar 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.
C1HierarchicalPresenter.Header	Gets or sets the item that labels the control.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1Toolbar control.

Property	Description
Padding	Gets or sets the padding inside a control. This is a dependency property.
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property..
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. 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 **C1Toolbar** 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.

C1Toolbar 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 Toolbar 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 C1Toolbar control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the [Template](#) property to customize the 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.

Additional Templates

In addition to the default template, the C1Toolbar control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1Toolbar control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**:

C1Toolbar Styles

ComponentOne Toolbar for Silverlight's Toolbar control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Toolbar ClearStyle Properties

Toolbar for Silverlight supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **C1Toolbar** elements. The supported properties for **C1Toolbar** are listed in the following table:

Property	Description
Background	Gets or sets the background used to fill the C1Toolbar .
MouseOverBrush	Gets or sets the brush used to highlight the control when the mouse is hovering over it.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when they are clicked.
FocusBrush	Gets or sets the brush of the control when the control is focused.

Toolbar for Silverlight Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studios.

Note: ComponentOne samples are also available at <http://helpcentral.componentone.com>.

C# Toolbar Samples

Sample	Description
C1ToolbarStrip	Shows a lightweight toolstrip control which can have vertical and horizontal orientation.
Toolbar	Ribbon-like toolbar control with groups.

Toolbar 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 **C1Toolbar** control in general. If you are unfamiliar with the **ComponentOne Toolbar for Silverlight** product, please see the [Toolbar for Silverlight Quickstart](#) (page 20) first.

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

Each task-based help topic also assumes that you have created a new WPF project. For additional information on this topic, see [Creating a .NET Project in Visual Studio](#) or [Creating a Microsoft Blend Project](#).

Aligning Toolbar Buttons

The following XAML shows how to align the toolbar buttons in the center of the group panel:

```
<c1:C1ToolbarGroup Header="Group" >
  <c1:C1ToolbarGroup.ItemsPanel>
    <ItemsPanelTemplate>
      <c1:C1ToolbarGroupPanel HorizontalAlignment="Center" />
    </ItemsPanelTemplate>
  </c1:C1ToolbarGroup.ItemsPanel>
```

Adding an Image to the Toolbar Button

You can use the `LargeImageSource` or `SmallImageSource` properties to add large or small images to your `C1ToolbarButtons`.

Using the Designer

1. Right-click the **C1ToolbarButton** control on your page and select **Properties**.
2. Locate the **LargeImageSource** property and click the **ellipsis** button next to it. The **Choose Image** dialog box appears.
3. Click the **Add** button and select the image you wish to add to it and click **OK**.

Adding Logic Behind the ToolbarButton Click Event

This topic shows how to add logic behind the `C1ToolbarButton` **Click** event for `C1ToolbarButtons` through XAML representation and then adding the code for the method in the Code Editor.

1. Add a **C1ToolbarButton** to the `C1Toolbar` element in the XAML editor and set the `LabelText` property to `Search`.

```
<c1:C1Toolbar Grid.Row="1" Height="100" Name="c1Toolbar1">
  <c1:C1ToolbarButton LabelTitle="Search"
```

```
</c1:C1Toolbar>
```

2. Add an attribute named **Click** to the **C1ToolbarButton** element in the XAML editor, and set its value to New Click. This is the name that you will give the event handler in code. Also give the **C1ToolbarButton** element a unique name and set its value to Search.

```
<c1:C1Toolbar Grid.Row="1" Height="100" Name="c1Toolbar1">  
  <c1:C1ToolbarButton LabelTitle="Search" Click="New_Click"  
  Name="Search"/>  
</c1:C1Toolbar>
```

3. Right-click the designer and then click **View Code**.
4. Add the following event handler to the Window1 class. This code displays a message whenever you click the button.

```
private void New_Click(object sender, RoutedEventArgs e)  
{  
  MessageBox.Show("Event handler was created by clicking the " + Search);  
}
```

Run the application and observe:

The name of the toolbar button appears in the message box.

Commanding with C1ToolBar for Silverlight

Silverlight 4 supports the **ICommand** interface on any object that derives from the **ButtonBase** class. Any button, including **C1ToolBarButton** , can be associated with an object that implements **ICommand** through the control's **Command** property. Although the **ICommand** interface is supported, Silverlight does not provide any built-in implementation. ComponentOne Studio for Silverlight includes a couple implementations, **C1Command** and **C1ToolBarCommand** , so you do not have to write one yourself.

The **C1Command** class is included in the **C1.Silverlight** assembly. **C1ToolBarCommand** , included in the **C1.Silverlight.Toolbar** assembly, extends **C1Command** by adding a few extra toolbar related properties for labels and images.

The following steps demonstrate how to use commanding with **C1ToolBar** .

Using C1ToolBarCommand

1. Open or create a new Silverlight application.
2. Add two **RowDefinitions** to the default **Grid** element, giving the first row an automatic height, such as this:

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

3. Add a **C1ToolBar** to fill the first row.
4. Add a **TextBox** in the second row.
5. Paste the following **C1ToolBarCommand** to the **UserControl Resources** such as this:

```
<UserControl.Resources>
  <c1:C1ToolBarCommand x:Key="cmdClear" LabelTitle="Clear Text"
  LargeImageSource="/Resources/delete.png" />
</UserControl.Resources>
```

This command will be used to clear the contents from the **TextBox** . You can specify a **LargeImageSource** and a **SmallImageSource** for **C1ToolBarCommand** , although this is not required.

6. Paste the following **XAML** to fill **C1ToolBar** with a tab, group and button:

```
<c1:C1ToolBar Name="c1ToolBar1">
  <c1:C1ToolBarTabControl>
    <c1:C1ToolBarTabItem Header="Home">
      <c1:C1ToolBarGroup Header="Application">
        <c1:C1ToolBarButton
  c1:CommandExtensions.Command="{StaticResource cmdClear}" />
      </c1:C1ToolBarGroup>
    </c1:C1ToolBarTabItem>
  </c1:C1ToolBarTabControl>
</c1:C1ToolBar>
```

Here we are setting the attached **c1:CommandExtensions.Command** property to our command. Note that **C1ToolBarButton** also has an inherited **Command** property. It is better to use the attached property **CommandExtensions.Command** to set the command when using **C1Commands** .

7. Add a **TextBox** to the page below the toolbar named "textBox1"

8. Next, you need to register the command after the page initializes with this code:

```
// register command methods
CommandManager.RegisterClassCommandBinding(GetType(), new
CommandBinding((C1ToolBarCommand)Resources["cmdClear"], Clear, CanClear));
// check the ability of registered commands to execute
CommandManager.InvalidateRequerySuggested();
```

The `C1.Silverlight.CommandManager` provides command related utility methods for registering commands. Here we pass event handlers for the `Clear` and `CanClear`

9. Paste the following `Clear` and `CanClear` event handlers which perform the logic for the command:

```
private void Clear(object sender, ExecutedRoutedEventArgs e)
{
    textBox1.Text = "";
}

private void CanClear(object sender, CanExecuteRoutedEventArgs e)
{
    if (textBox1.Text.Length > 0)
    {
        e.CanExecute = true;
    }
    else
        e.CanExecute = false;
}
```

10. In Silverlight you need to explicitly check the ability of registered commands in code. In the `TextBox.TextChanged` event, add the following code:

```
private void textBox1_TextChanged(object sender, TextChangedEventArgs e)
{
    CommandManager.InvalidateRequerySuggested();
}
```

11. Run the sample and you can now clear the contents of the textbox using a command in `C1ToolBar` using `C1Command`.

