ComponentOne

TabControl for WPF

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 TabControl for WPF Overview 1
- What's New in TabControl for WPF 1
- Installing TabControl for WPF 1
  - TabControl for WPF Setup Files 1
  - System Requirements 2
  - Installing Demonstration Versions 3
  - Uninstalling TabControl for WPF 3
- End-User License Agreement 3
- Licensing FAQs 3
  - What is Licensing? 3
  - How does Licensing Work? 4
  - Common Scenarios 4
  - Troubleshooting 6
- Technical Support 8
- Redistributable Files 9
- About this Documentation 9
- XAML and XAML Namespaces 9
- Creating a Microsoft Blend Project 10
- Creating a .NET Project in Visual Studio 11
- Creating an XAML Browser Application (XBAP) in Visual Studio 12
- Adding the TabControl for WPF Components to a Blend Project 13
- Adding the TabControl for WPF Components to a Visual Studio Project 14

Key Features 15

TabControl for WPF Quick Start 16
- Step 1 of 4: Creating an Application with a C1TabControl Control 16
- Step 2 of 4: Adding Tab Pages to the C1TabControl Control 17
- Step 3 of 4: Customizing the C1TabControl Control 18
- Step 4 of 4: Running the Project 18

Working with C1TabControl 20
- C1TabControl Elements 20

Tabs 21
Tabstrip 21
Tab Page 22
C1TabControl Features 23
Tab Shaping 23
Tabstrip Placement 23
Tab Closing 24
Optional Tab Menu 25
Tab Overlapping 25
TabControl for WPF Layout and Appearance 26
ComponentOne ClearStyle Technology 26
How ClearStyle Works 26
C1TabControl ClearStyle Properties 27
TabControl for WPF Appearance Properties 28
TabControl Templates 30
Item Templates 31
TabControl for WPF Samples 31
TabControl for WPF Task-Based Help 31
Adding a Tab to the C1TabControl Control 31
Adding Content to a Tab Page 32
Specifying a Tab Header 34
Changing the Tabstrip Placement 34
Changing the Shape of Tabs 35
Allowing Users to Close Tabs 36
Preventing a User from Closing a Specific Tab 37
Adding a Menu to the Tabstrip 38
Overlapping Tabs on a Tabstrip 39

ComponentOne TabControl for WPF Overview

Arrange content in an efficient, organized manner using **ComponentOne TabControl for WPF.** The **C1TabControl** control allows you to add tabs and corresponding content pages, thus enabling you to dispense substantial amounts of information while reducing screen space usage.

Getting Started

- Working with C1TabControl (page 20)
- Quick Start (page 16)
- Task-Based Help (page 31)

What's New in TabControl for WPF

The **TabControl for WPF** documentation was last updated on October 29, 2010 for its 2010 v3 release.

The following features have been added to **TabControl for WPF**:

- **TabControl for WPF** now features ComponentOne's ClearStyle technology, a new and simple approach to providing Silverlight and WPF control styling. For more information about ClearStyle technology, see the ComponentOne ClearStyle Technology (page 26) topic.
- You can now use a unified namespace, `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"`, in your WPF projects. All ComponentOne WPF controls will be accessible through XAML using this namespace. Please note, however, that you must add a reference to the assembly of each control you wish to use.

Tip: A version history containing a list of new features, improvements, fixes, and changes for each product is available in HelpCentral at http://helpcentral.componentone.com/VersionHistory.aspx.

Installing TabControl for WPF

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

TabControl for WPF Setup Files

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

Bin

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

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

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

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

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

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

C1WPF\XAML	Contains the full XAML definitions of C1TabControl styles and templates which can be used for creating your own custom styles and templates.

The **ComponentOne Studio for WPF Help Setup** program installs integrated Microsoft Help 2.0 and Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for WPF directory in the following folders:

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.

Samples

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

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

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

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

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

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

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2) Windows Vista™ Windows 2008 Server Windows 7
Environments:	.NET Framework 3.5 or later

	Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP Visual Studio® 2008 or later
Microsoft® Expression® Blend Compatibility:	**TabControl for WPF** includes design-time support for Expression Blend.

Note: The **C1.WPF.VisualStudio.Design.dll** assembly is required by Visual Studio 2008 and the **C1.WPF.Expression.Design.dll** assembly is required by Expression Blend. The **C1.WPF.Expression.Design.dll** and **C1.WPF.VisualStudio.Design.dll** assemblies installed with **TabControl for WPF** should always be placed in the same folder as **C1.WPF.dll**; the DLLs should NOT be placed in the Global Assembly Cache (GAC).

Installing Demonstration Versions

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

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

Uninstalling TabControl for WPF

To uninstall **ComponentOne TabControl for WPF**:

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

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

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

End-User License Agreement

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

Licensing FAQs

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

What is Licensing?

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

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

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

How does Licensing Work?

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

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

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

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

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

These files are automatically added to the project.

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

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

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

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

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

Common Scenarios

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

Creating components at design time

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

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

Creating components at run time

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

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

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

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

Inheriting from licensed components

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

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

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

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

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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

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

Using licensed components with automated testing products

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

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

For example:

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

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

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

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

Troubleshooting

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

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

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

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

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

If that fails follow these steps:

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

Alternatively, follow these steps:

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

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

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

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. 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 TabControl for WPF is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.dll

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

- WPFToolkit.dll

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

About this Documentation

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.

XAML and XAML Namespaces

XAML is a declarative XML-based language that is used as a user interface markup language in Windows Presentation Foundation (WPF) and the .NET Framework 3.0. With XAML you can create a graphically rich customized user interface, perform data binding, and much more. For more information on XAML and the .NET Framework 3.0, please see http://www.microsoft.com.

XAML Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

When you create a Microsoft Expression Blend project, a XAML file is created for you and some initial namespaces are specified:

Namespace	Description
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"	This is the default Windows Presentation Foundation namespace.
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"	This is a XAML namespace that is mapped to the **x:** prefix. The **x:** prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications.

When you add a C1TabControl control to the window in Microsoft Expression Blend or Visual Studio, **Blend** or **Visual Studio** automatically creates an XML namespace for the control. The namespace looks like the following:

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

The namespace value is **c1**. This is a unified namespace; once this is in the project, all ComponentOne WPF controls found in your references will be accessible through XAML (and Intellisense). Note that you still need to add references to the assemblies for each control you need to use.

You can also choose to create your own custom name for the namespace. For example:

```
xmlns:MyMTB="clr-namespace:C1.WPF;assembly=C1.WPF">
```

You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the panel:

```
<MyMTB:C1TabControl Name="c1TabControl1" BorderThickness="10,10,10,10">
```

Creating a Microsoft Blend Project

To create a new Blend project, complete the following steps:

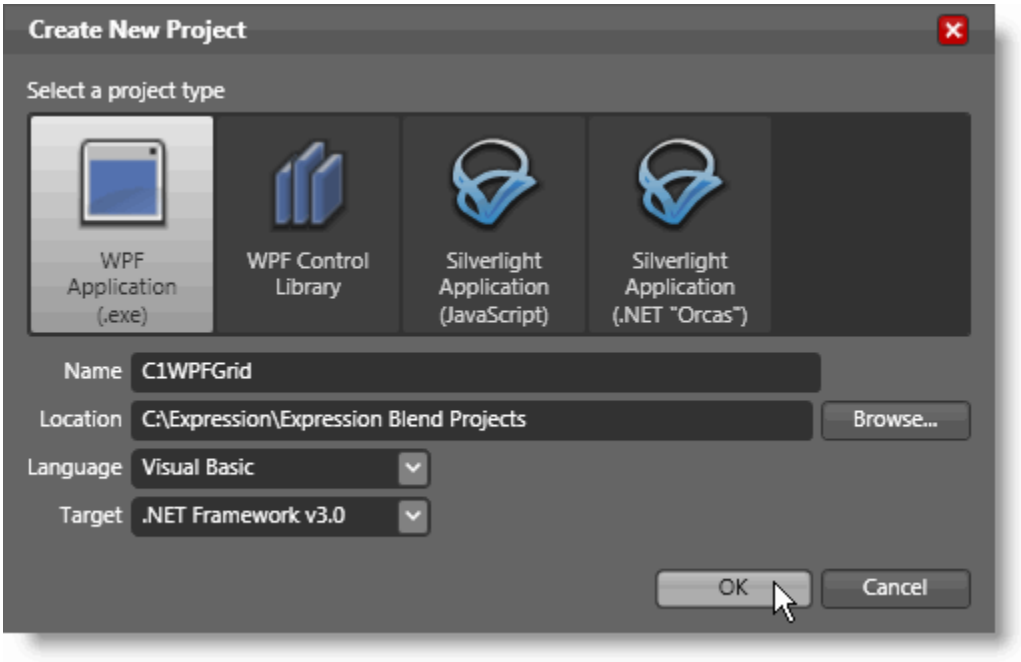
1. From the **File** menu, select **New Project** or click **New Project** in the Blend startup window.

   The **Create New Project** dialog box opens.

2. Make sure **WPF Application (.exe)** is selected and enter a name for the project in the Name text box. The **WPF Application (.exe)** creates a project for a Windows-based application that can be built and run while being designed.

3. Select the **Browse** button to specify a location for the project.

4. Select a language from the **Language** drop-down box and click **OK**.



A new Blend project with a XAML window is created.

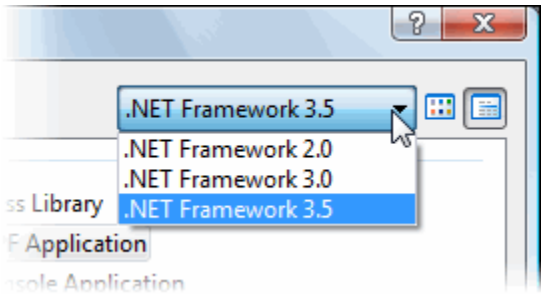
Creating a .NET Project in Visual Studio

To create a new .NET project in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.

   The **New Project** dialog box opens.

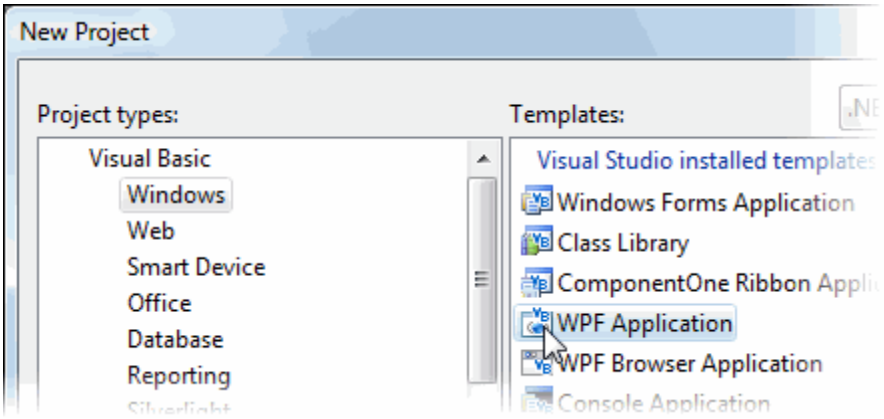
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



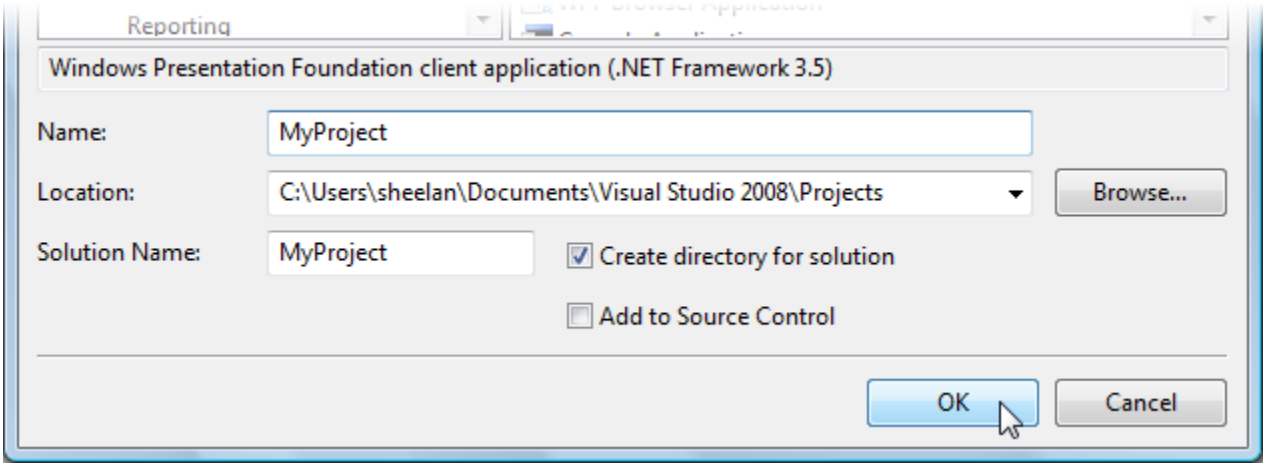
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

   Note: In Visual Studio 2005 select **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the Project types menu.

4. Choose **WPF Application** from the list of **Templates** in the right pane.



5. Enter a name for your application in the **Name** field and click **OK**.



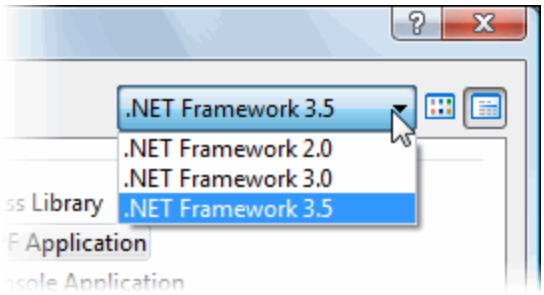
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

Note: You can create your WPF 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, Blend will be used for most examples.

Creating an XAML Browser Application (XBAP) in Visual Studio

To create a new XAML Browser Application (XBAP) in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**. The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



3. Under Project types, select either **Visual Basic** or **Visual C#**.
4. Choose **WPF Browser Application** from the list of **Templates** in the right pane.

> **Note:** If using Visual Studio 2005, you may need to select **XAML Browser Application (WPF)** after selecting **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the left-side menu.

5. Enter a name for your application in the **Name** field and click **OK**.

   A new Microsoft Visual Studio .NET WPF Browser Application project is created with a XAML file that will be used to define your user interface and commands in the application.

Adding the TabControl for WPF Components to a Blend Project

In order to use C1TabControl or another **ComponentOne TabControl for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF.Extended** assembly and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

1. Select **Project** | **Add Reference**.
2. Browse to find the **C1.WPF. dll** assembly installed with **TabControl for WPF**.

> **Note:** The **C1.WPF.dll** file is installed to **C:\Program Files\ComponentOne Studio .NET 3.0\bin** by default.

3. Select **C1.WPF. 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.WPF.Extended** assembly, click the **Asset Library** button  in the Blend Toolbox. The **Asset Library** appears:

2. Click the **Custom Controls** tab. All of the **TabControl for WPF** main and auxiliary components are listed here.
3. Select **C1TabControl**. The component will appear in the Toolbox above the **Asset Library** button.
4. Double-click the **C1TabControl** component in the Toolbox to add it to **Window1.xaml**.

Adding the TabControl for WPF Components to a Visual Studio Project

When you install **ComponentOne TabControl for WPF** the C1TabControl control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne TabControl for WPF provides the following control:

- C1TabControl

To use a **TabControl for WPF** panel or control, add it to the window or add a reference to the **C1.WPF** assembly to your project.

Manually Adding TabControl for WPF to the Toolbox

When you install **TabControl for WPF**, the following **TabControl for WPF** control and panel will appear in the Visual Studio Toolbox customization dialog box:

- C1TabControl

To manually add the C1TabControl 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 **TabControl for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFTabControl**, 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 **WPF Components** tab.

5. Sort the list by Namespace (click the *Namespace* column header) and select the check boxes for components belonging to the **C1.WPF.Extended** namespace. Note that there may be more than one component for each namespace.

Adding TabControl for WPF to the Window

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

1. Add the C1TabControl control to the Visual Studio Toolbox.
2. Double-click C1TabControl or drag the control onto the window.

Adding a Reference to the Assembly

To add a reference to the **TabControl for WPF** assembly, complete the following steps:

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

This makes the objects defined in the **TabControl for WPF** assembly visible to the project.

Key Features

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

- **Scroll Elements**

  The C1TabControl control will automatically insert scroll buttons when the amount of tabs exceeds the specified width or height of the control.

- **Tabstrip Placement**

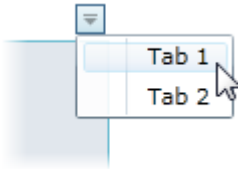
  The C1TabControl control's can be placed at the top, bottom, left, or right of the control.

- **Tab Closing Options**

  Control whether the user can close tabs and where to show the close button. Display the close button inside each tab item or in a global location outside the tabstrip, just like Visual Studio does in its Documents tab.

- **Show Tabs in a Menu**

  The C1TabControl control has an optional tab menu that can be activated by setting the **TabStripMenuVisibility** property to **Visible**. The tab menu enables users to open tab pages using a drop-down menu.



- **Customize the Header's Shape**

  You can modify the shape of the tab headers using the TabItemShape property in the TabControl; select from **Rounded**, **Rectangle**, and **Sloped**. This is ideal for non-designer users; you don't need to change the template of the control to change the shape of the tab headers.

- **New Tab Item**

  C1TabControl includes built-in support for the new tab item with a uniform look and feel with the rest of the tabs, just like Microsoft Internet Explorer 8.

- **Align Items in the Header**

  Define if the tab items are overlapped with the right-most in the back or the left-most in the back. The selected item is always on top.

- **Change the Background**

  Keeping the appearance of the tab, you can change its background. While this seems like a very simple feature, C1TabControl is the only tab control in the market that allows you to change the background without having to customize the full template.

- **Overlap Headers**

  Overlap between tab items headers can be customized to show jagged tabs, like the Documents tab in Microsoft Visual Studio.

TabControl for WPF Quick Start

The following quick start guide is intended to get you up and running with **TabControl for WPF**. In this quick start, you'll start in Blend to create a new project with the C1TabControl control. You will also customize the C1TabControl control, add tabs pages filled with content, and then observe some of the run-time features of the control.

Step 1 of 4: Creating an Application with a C1TabControl Control

In this step, you'll begin in Visual Studio to create a WPF application using **TabControl for WPF**.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select **WPF Application**.
3. Enter a **Name** and **Location** for your project and click **OK** to create the new application.
4. In the Toolbox, double-click the C1TabControl icon to add the C1TabControl control to the WPF application.
5. Add three tabs to the control by completing the following steps:
   a. Click the C1TabControl control once to select it.
   b. In the **Properties** window, click the **Items** ellipsis button  .

The **Collection Editor: Items** dialog box opens.

c. Click the **Add** button three times to add three C1TabItem items to the C1TabControl control.

6. Click **OK** to close the **Collection Editor: Items** dialog box.

You have completed the first step of the **TabControl for WPF** quick start. In this step, you created a project and added a C1TabControl control with three tabs to the project. In the next step, you will customize the control's tab pages.

Step 2 of 4: Adding Tab Pages to the C1TabControl Control

In the last step, you created a WPF project in Expression Blend and then added a C1TabControl control with three tabs to it. In this step, you will customize each tab page.

Complete the following steps:

1. Switch to XAML view.
2. Add `Header="Tab 1"` to the first `<c1:C1TabItem>` tag so that the markup resembles the following:

```
<c1:C1TabItem Header="Tab 1"/>
```

3. Add `Header="Tab 2"` and `Content="I am the Content property set to a string"` to the second `<c1:C1TabItem>` tag so that the markup resembles the following:

```
<c1:C1TabItem Header="Tab 2" Content="I am the Content property set to a
string"/>
```

4. Add `Header="Tab 3"`, `Content="You can't close this tab. Try it.`", and `CanUserClose="False"` to the third `<c1:C1TabItem>` tag so that the markup resembles the following:

```
<c1:C1TabItem Header="Tab 3" Content="You can't closet his tab. Try it."
CanUserClose="False"/>
```

Setting the CanUserClose property to **False** prevents users from closing the tab at run time.

5. Add a **Calendar** control as the first tab's by completing the following steps:
   a. Switch to Design view and select the first tab.
   b. In the Toolbox, double-click the **Calendar** icon to add the **Calendar** control to the tab.
   c. Select the **Calendar** control and then set the following properties:
      - Set the **Height** property to "Auto".
      - Set the **Width** property to "Auto".

You have completed step 2 of 4. In this step, you customized the three pages of the C1TabControl control. In the next step, you'll customize the appearance and behavior of the C1TabControl control.

Step 3 of 4: Customizing the C1TabControl Control

In the last step, you added three customized tab pages to the C1TabControl control. In this step, you'll customize the C1TabControl control by setting several of its properties.

Complete the following steps:

1. Select the C1TabControl control.
2. In the **Properties** window, set the following properties:
   - Set the **Height** property to "200".
   - Set the **Width** property to "300".
   - Set the TabItemClose property to **InEachTab**. This will add close buttons to each tab except to the third tab, which you specified shouldn't be allowed to close in the last step of the quick start.
   - Set the TabItemShape property to **Sloped**. This will change the shape of the tab items so that they resemble tabs on an office folder.
   - Set the TabStripMenuVisibility property to **Visible**.
   - Set the TabStripPlacement property to **Bottom**. This will place the tabstrip at the bottom of the control.

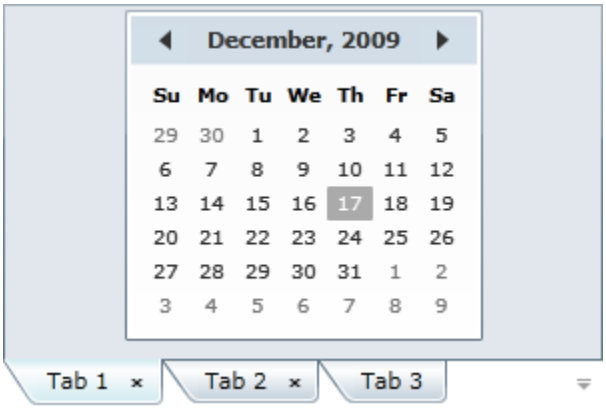
You have completed step 2 of 4 by customizing the features of the C1TabControl control. In the next step, you will run the program and observe what you've accomplished during this quick start.

Step 4 of 4: Running the Project

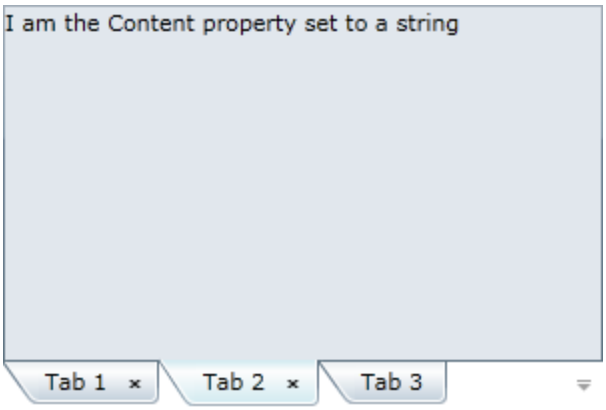
In the previous steps, you created a project with a C1TabControl control, added tab pages to the control, and modified the control's appearance and behaviors. In this step, you will run the program and observe all of the changes you made to the C1TabControl control.

Complete the following steps:

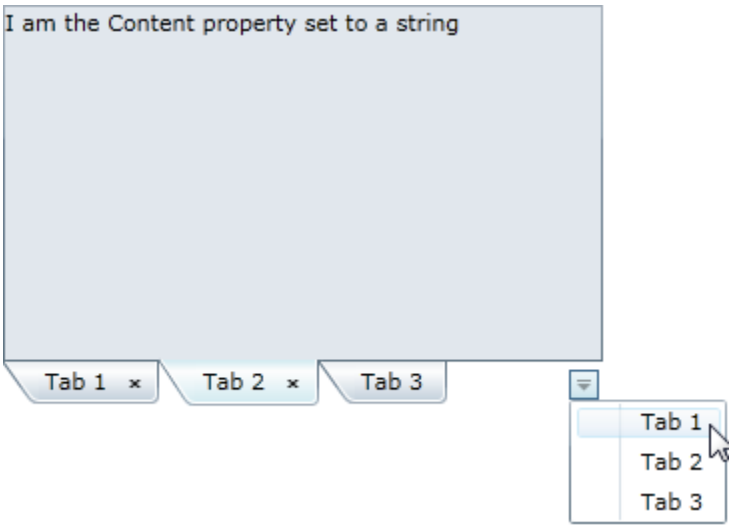
1. Press F5 to run the project. Observe that the C1TabControl control's tabstrip runs along the bottom of the control and features sloped tabs. Also note that it loads with the first tab page, which features a **Calendar** control as content, in view.



2. Click **Tab 2** and observe that the content is just the **Content** property set to a string.

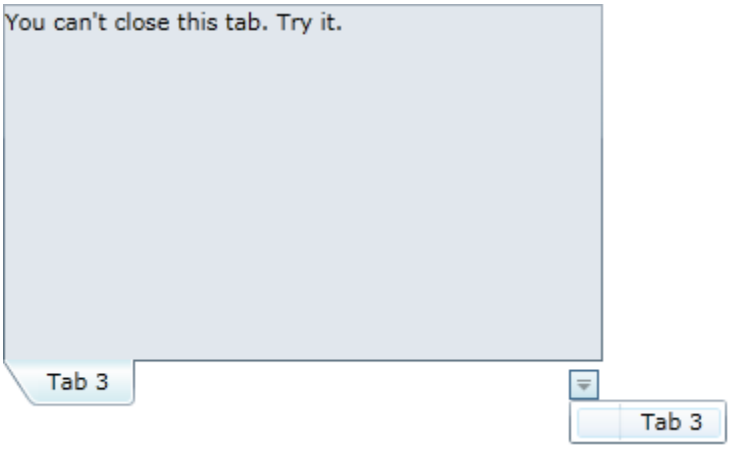


3. Click the drop-down button to open the tab menu and select **Tab 1**.



Tab 1 takes focus again.

4. Click **Tab 1**'s close button to close the tab.
5. Click **Tab 2**'s close button to close the tab. **Tab 3**'s content comes into focus; you can't close this tab because you set its CanUserClose property to **False** in a previous step.
6. Click the menu button and note that only the current tab page, **Tab 3**, is listed because the other two are closed.

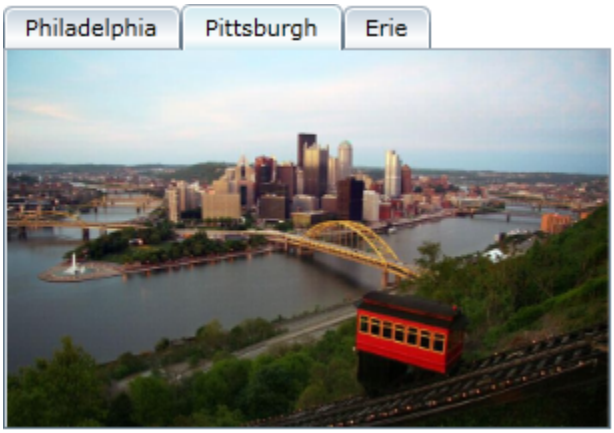


Congratulations! You have completed all four steps of the **TabControl for WPF** quick start. In this quick start, you created a project with a fully customized C1TabControl. From here, you can visit Working with C1TabControl (page 20) to learn more about the control's essentials or visit TabControl for WPF Task-Based Help (page 31) to jump right into using the control.

Working with C1TabControl

The C1TabControl control is a container that that can hold a series of tab pages. Each tab page can store text, images, and controls. Each tab and tab page is represented by the C1TabItem class.

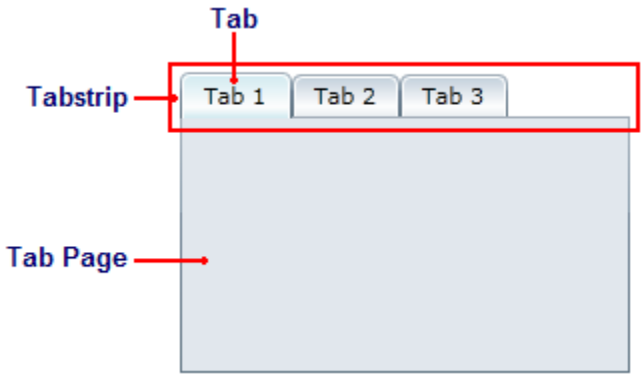
When you add the C1TabControl control to a project, it exists as nothing more than a container. But once the control is added to your project, you can easily add tabs to it in Design view, in XAML, or in code. The following image depicts a C1TabControl control with three tabs.



The following topics provide an overview of the C1TabControl control's elements and features.

C1TabControl Elements

This section provides a visual and descriptive overview of the elements that comprise the C1TabControl control. The control is comprised of three elements – the tab, the tabstrip and the tab page – that combine to make the complete C1TabControl control.



The topics below describe each element of the C1TabControl control.

Tabs

Each tab on the C1TabControl control is represented by the C1TabItem class. The header of the tab is exposed by the Header property of the C1TabItem. Each tab is associated with a tab page (see Tab Page (page 22)).

Tabs can appear rounded, sloped, or rectangular. When a tab is added to a page, its default appearance is sloped – it looks similar to a tab on an office folder.

You can add a close button to each tab by setting the TabItemClose property to **InEachTab**. This will allow users to close any tab in the control; however, you can deny users the ability to close a particular tab by setting that tab's CanUserClose property to **False**.

Customizing the Header

When you want to add something simple to the tab's header, such as an unformatted string, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1TabItem Header="Hello World"/>
```

You can also customize the header using the HeaderTemplate.

Tabstrip

The C1TabControl control's tabstrip is created by adding one or more C1TabItem items to the control. Each tab in the strip is associated with a tab page (see Tabs (page 21) and Tab Page (page 22)).

By default, the tabstrip appears along the top of the C1TabControl control, but you can adjust the position of the strip by setting the TabStripPlacement property You can also adjust the overlap of the tabs by setting the TabStripOverlap and TabStripOverlapDirection properties.

The tabstrip can also contain two optional items: a close button and a menu drop-down arrow. The close button, when clicked, closes the selected tab; the menu drop-down arrow, when clicked, exposes a drop-down menu from which users can open a different tab.

To learn more about the close button, see Tab Closing (page 24). To learn more about the menu drop-down, see Optional Tab Menu (page 25).

Tab Page

When a tab (C1TabItem) is added to a C1TabControl control, it will have a corresponding tab page that will initially appear as an empty space. In the tab page, you can add grids, text, images, and arbitrary controls. When working in Blend or Visual Studio's Design view, you can add elements to the tab page using a simple drag-and-drop operation.

You can add text to the tab page by setting the item's **Content** property or by adding a **TextBox** element to the tab page. Adding elements to the tab page at run time is simple: You can either use simple drag-and-drop operations or XAML in Visual Studio or Blend. If you'd prefer to add a control at run time, you can use C# or Visual Basic code.

A C1TabItem item can only accept one child element at a time. However, you can circumvent this issue by adding a panel-based control as its child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the C1TabItem item, but its ability to hold multiple elements will allow you to show several controls in the tab page.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the tab page, such as an unformatted string or a single control, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1TabItem Content="Hello World"/>
```

However, there may be times where you want to add more complex elements, such as grids or panels, to the tab page. In this case you can use property element syntax, such as in the following:

```
<c1:C1TabItem>
    <c1:C1TabItem.Content>
        <StackPanel>
            <TextBlock Text="Hello"/>
            <TextBlock Text="World"/>
        </StackPanel>
    </c1:C1TabItem.Content>
</c1:C1TabItem>
```

You can also customize the content using the **ContentTemplate**.

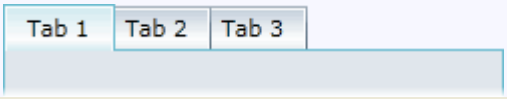
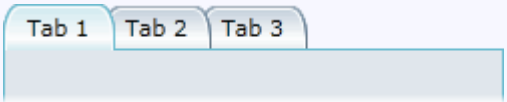
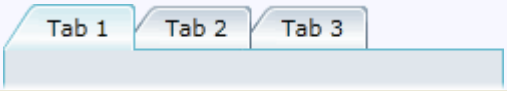
C1TabControl Features

This section details several important features of the C1TabControl control.

Tab Shaping

Tabs can appear rounded, sloped, or rectangular. By default, the tab will appear rounded, but you can change the shape of the tabs to any of the three settings by setting the **C1TabControl**.TabItemShape property to **Rectangle**, **Rounded**, or **Sloped**.

The following table illustrates each tab shape.

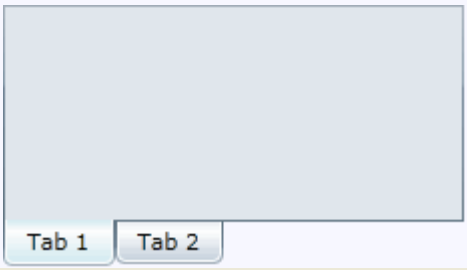
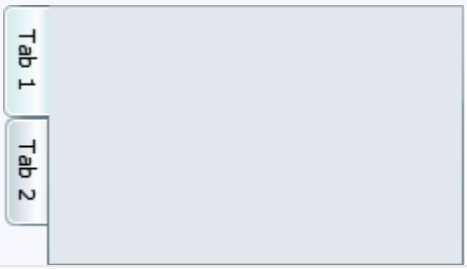
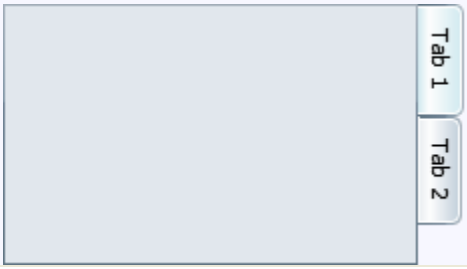
Tab Shape	Illustration
Rectangle	
Rounded	
Sloped	

For task-based help, see Changing the Shape of Tabs (page 35).

Tabstrip Placement

The C1TabControl control's tabstrip, by default, will appear along the top of the control. However, you can set the **C1TabControl**.TabStripPlacement property to **Bottom**, **Left**, or **Right** to change the position of the tabstrip.

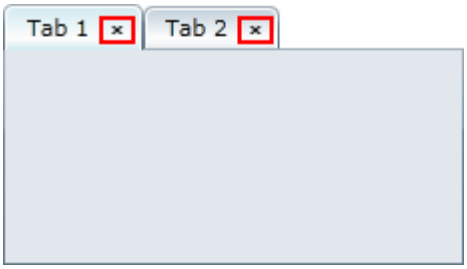
The following table illustrates each **C1TabControl**.TabStripPlacement setting.

ExpandDirection	Result
Top	
Bottom	
Left	
Right	

For task-based help, see Changing the Tabstrip placement (page 34).

Tab Closing

You can add a close button to each tab by setting the TabItemClose property to **InEachTab**. This will allow users to close any tab in the control. On-tab close buttons appear as follows:

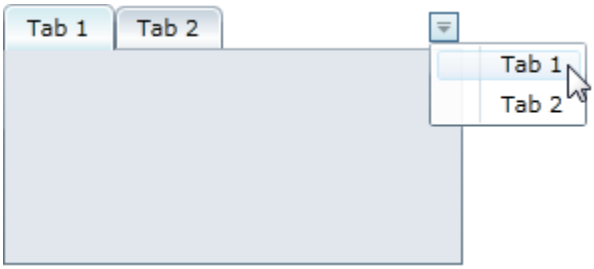


If you prefer, you can create a global close button that will appear on the tabstrip instead of directly on the tabs. To add the global close button, you set the TabItemClose property to **GlobalClose**. Clicking the global close button will close the currently selected tab. The global close button appears as follows:

If you want to prevent a user from closing a particular tab, just set that tab's CanUserClose property to **False**. For task-based help about allowing tab closing, see Allowing Users to Close Tabs (page 36); for task-based help about preventing tab closure, see Preventing a User from Closing a Specific Tab (page 37).

Optional Tab Menu

The C1TabControl control allows you to add a drop-down menu that allows users to select a tab and tab page from a menu. When enabled, this drop-down menu is accessible from the control's tabstrip. The drop-down menu appears as follows:

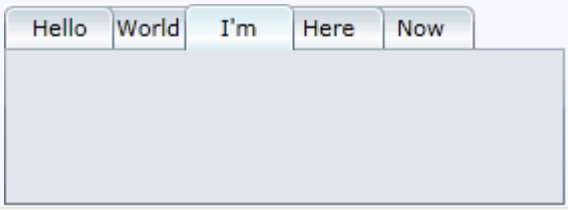
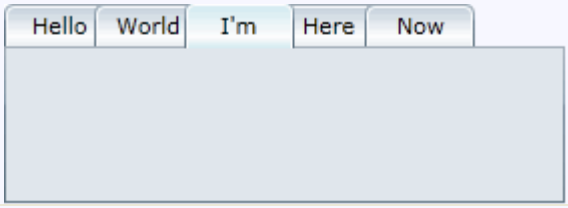
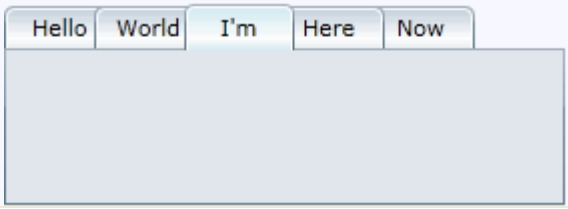


To activate the drop-down menu, simply set the **C1TabControl**.TabStripMenuVisibility property to **Visible**. For task-based help about adding a tab menu, see Adding a Menu to the Tabstrip (page 38).

Tab Overlapping

You can control the overlap of tabs by setting the TabStripOverlap property and the TabStripOverlapDirection property (for task-based help, see Overlapping Tabs on a Tabstrip (page 39)). The TabStripOverlap property controls how many pixels of the tab overlap, and the TabStripOverlapDirection direction property allows you to select an enumeration value that sets the direction of the overlap. The TabStripOverlapDirection property has three enumeration values – **Right**, **Left**, and **RightLeftFromSelected** – which are described and illustrated in the table below. Please note that the TabStripOverlap property for each of the following examples has been set to a value of 10.

Enumeration Value	Description	Illustration

Right	The rightmost tabs are in the back while the selected tab is in the front.	
Left	The leftmost tabs are in the back while the selected tab is in the front.	
RightLeftFromSelected	Leftmost tabs are in the back, rightmost tabs are in the back, and the selected tab is in the front.	

TabControl for WPF Layout and Appearance

The following topics detail how to customize the C1TabControl control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

ComponentOne ClearStyle Technology

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

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

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

How ClearStyle Works

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

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

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

C1TabControl ClearStyle Properties

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

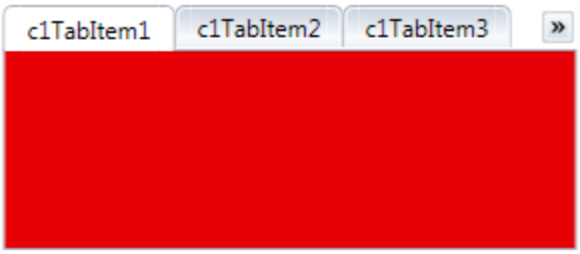
The following table outlines the brush properties of the **C1TabControl** control:

Brush	Description
Background	Gets or sets the brush for the background of each **C1TabItem**'s content area.
TabStripBackground	Gets or sets the brush for the background of the **C1TabStrip**'s background.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tabs when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tabs when they are clicked on.

The following table outlines the brush properties of the **C1TabItem** control:

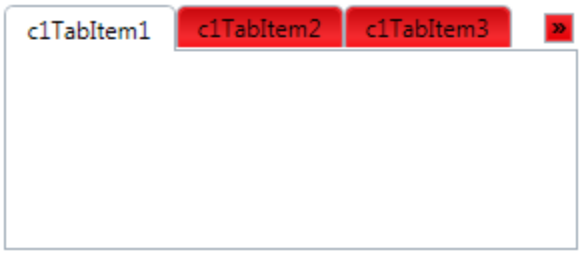
Brush	Description
Background	Gets or sets the backgrounds of all **C1TabItem**s associated with the control.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tab when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tab when they are clicked on.

You can completely change the appearance of the **C1TabControl** control by setting a few properties, such as the **Background** property, which sets the background color of the tab control's content area. For example, if you set the **Background** property to "#FFE40005", the **C1TabControl** control would appear similar to the following:



If you clicked through the tabs, you'd notice that the background of each tab's content area is red. If you'd like to change the color of one tab, simply set the **C1TabItem**'s **Background** property rather than the **C1TabControl**'s **Background** property.

You can also set the color of the tabstrip by setting the **C1TabControl**'s **TabStripBackground** property. In the following example, the **TabStripBackground** property is set to "#FFE40005":



It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology (page 26) topic.

TabControl for WPF Appearance Properties

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

Text Properties

The following properties let you customize the appearance of text in the C1TabControl control and its items.

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.
TextAlignment	Gets or sets how the text should be aligned in the tab. This is a dependency property.
Header	Gets or sets the header of a tab item.
HeaderFontFamily	Gets or sets the font family of the header.
HeaderFontStretch	Gets or sets the font stretch of the header.
HeaderFontStyle	Gets or sets the font style of the header.
HeaderFontWeight	Gets or sets the font weight of the header.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1TabControl control and its items.

Property	Description
HeaderPadding	Gets or sets the padding of the header.
HeaderHorizontalContentAlignment	HorizontalContentAlignment of the header.
HeaderVerticalContentAlignment	Gets or sets the vertical content alignment of the header.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. 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 C1TabControl control and its items.

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.
HeaderBackground	Gets or sets the background brush of the header.
HeaderForeground	Gets or sets the foreground brush of the header.

Border Properties

The following properties let you customize the border of the C1TabControl control and its items.

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 C1TabControl control and its items.

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

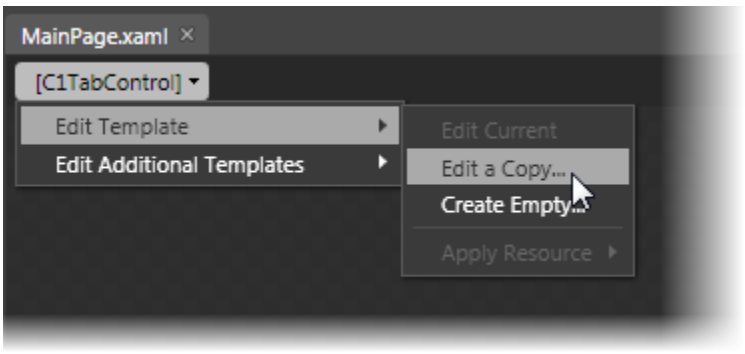
	element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element.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

TabControl Templates

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

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1TabControl control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



If you want to edit the C1TabItem template, simply select the C1TabItem control and, in the menu, select **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

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

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

Additional Templates

In addition templates, the C1TabControl control and C1TabItem control include a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1TabControl or C1TabItem control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

Item Templates

ComponentOne TabControl for WPF's tab control is an **ItemsControl**s that serves as a container for other elements. As such, the control includes templates to customize items places within the tab control. These templates include an **ItemTemplate**, an **ItemsPanel**, and an **ItemContainerStyle** template. You use the **ItemTemplate** to specify the visualization of the data objects, the **ItemsPanel** to define the panel that controls the layout of items, and the **ItemStyleContainer** to set the style of all container items.

Accessing Templates

You can access these templates in Microsoft Expression Blend by selecting the C1TabControl control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit Generated Items (ItemTemplate)**, **Edit Layout of Items (ItemsPanel)**, or **Edit Generated Item Container (ItemStyleContainer)** and select **Create Empty** to create a new blank template or **Edit a Copy**.

A dialog box will appear allowing you to name the template and determine where to define the template.

TabControl for WPF Samples

Please be advised that these ComponentOne software tools are accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studios.

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

The following pages within the **ControlExplorer** detail the C1TabControl control:

Sample	Description
TabControl	Illustrates the functionality of the C1TabControl control.

TabControl for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1TabControl control in general. If you are unfamiliar with the **ComponentOne TabControl for WPF** product, please see the **TabControl for WPF** Quick Start first.

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

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

Adding a Tab to the C1TabControl Control

In this topic, you will add tabs to the C1TabControl control in Design view, in XAML, and in code.

In Design View in Blend

Complete the following steps:

1. Click the C1TabControl control once to select it.
2. In the Toolbox, double-click the C1TabItem icon to add a tab item to the C1TabControl control.

   A C1TabItem appears within the C1TabControl control.

In XAML

To add a tab to the C1TabControl control, place the following markup between the <c1:C1TabControl> and </c1:C1TabControl> tags:

```
<c1:C1TabItem Content="c1TabItem">
</c1ext:C1TabItem>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Import the following namespace:
   - Visual Basic
     ```
     Imports C1.WPF
     ```
   - C#
     ```
     using C1.WPF;
     ```
3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     'Create the tab and add content
     Dim C1TabItem1 As New C1TabItem()
     C1TabItem1.Content = "C1TabItem1"
     'Add the tab to the control
     c1TabControl1.Items.Add(C1TabItem1)
     ```
   - C#
     ```
     //Create the tab and add content
     c1TabItem c1TabItem1 = new c1TabItem();
     c1TabItem1.Content = "c1TabItem1";
     //Add the tab to the control
     c1TabControl1.Items.Add(c1TabItem1);
     ```
4. Run the program.

Adding Content to a Tab Page

You can add content to the tab page using the **Content** property. This topic demonstrates how to add content – in this case, a standard **Button** control – to a tab page in Design view, in XAML, and in code. This topic assumes that you have added at least one C1TabItem to the C1TabControl control.

In Design View

Complete the following steps:

1. In Design view, select the tab you wish to add the content to.
2. In the Toolbox, double-click the **Button** icon to add a **Button** control to the tab.
3. Double click the **Button** icon to add it to the tab page's content area.
4. Select the Button control and set the following properties in the **Properties** window:

- Set the **Width** property to "Auto".
- Set the **Height** property to "Auto".

5. Run the program and click the tab.

In XAML

To add a **Button** control to the content area in XAML, place the following markup between the `<c1:C1TabItem>` and `</c1:C1TabItem>` tags:

```
<Button Content="Button" Height="Auto" Width="Auto"/>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = Double.NaN
NewButton.Height = Double.NaN
'Add the Button to the content area
c1TabItem1.Content = (NewButton)
```

   - C#

```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
//Add the Button to the content area
c1TabItem1.Content = (NewButton);
```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabItem with a **Button** control as content.

Specifying a Tab Header

In this topic, you will add a header to a tab by setting the **Header** property Blend, in XAML, and in code. This topic assumes that you have added at least one C1TabItem to the C1TabControl control.

In XAML

To add a header to a tab, add `Header="Hello World"` to the `<c1:C1TabItem>` tag so that the markup resembles the following:

```
<c1:C1TabItem Header="Hello World"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabItem1.Header = Hello World
     ```

   - C#

     ```
     c1TabItem1.Header = Hello World;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabItem with a header.

Changing the Tabstrip Placement

The tabstrip of a C1TabControl control is placed on the top by default, but you can also place it to the left, right, or bottom of the control by setting the TabStripPlacement property. In this topic, you will set the TabStripPlacement property to **Right** in Design view, in XAML, and in code. For more information, see Tabstrip Placement (page 23). For more information on tabstrip placement settings, see Tabstrip Placement (page 23).

In Design View

Complete the following steps:

1. Select the C1TabControl control.
2. In the **Properties** window, click the TabStripPlacement drop-down arrow and select **Right** from the list.

In XAML

To change the tabstrip placement, add `TabStripPlacement="Right"` to the `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl TabStripPlacement="Right"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabControl1.TabStripPlacement = Right
     ```

   - C#

     ```
     c1TabControl1.TabStripPlacement = Right;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control with its tabstrip placed on its right side.

Changing the Shape of Tabs

The tabs of a C1TabControl control can be rectangular, rounded, or sloped. By default, the tabs are rounded, but you can change the shape of the tabs by setting the C1TabControl control's TabItemShape property. In this topic, you will change the shape of the tabs to **Sloped** in Design view, in XAML, and in code. For more information on tabstrip shaping settings, see Tab Shaping (page 23).

In Design View

Complete the following steps:

1. Select the C1TabControl control.
2. In the **Properties** window, click the TabItemShape drop-down arrow and select **Sloped** from the list.

In XAML

To change the shape of the tabs, add `TabItemShape="Sloped"` to the `<c1:C1TabControl>` tag so that the markup resembles the following:

```
<c1:C1TabControl TabItemShape="Sloped"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabControl1.TabItemShape = Sloped
     ```

   - C#

     ```
     c1TabControl1.TabItemShape = Sloped;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control with sloped tabs.

Allowing Users to Close Tabs

By default, users can't close the tabs on a C1TabControl control. You can alter this by setting the TabItemClose property to **InEachTab** or **GlobalClose** so that users can close tabs by clicking a button inside the tab or by selecting a tab and then clicking a universal close button (see Tab Closing (page 24) for more information). In this topic, you will set the TabItemClose property to **GlobalClose** in Design view, in XAML, and in code.

In Design View

Complete the following steps:

1. Select the C1TabControl control.
2. In the **Properties** window, click the TabItemClose drop-down arrow and select **GlobalClose** from the list.

In XAML

To allow users to close tabs using a universal button, add `TabItemClose="GlobalClose"` to `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl TabItemClose="GlobalClose"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabControl1.TabItemClose = GlobalClose
     ```

   - C#

     ```
     c1TabControl1.TabItemClose = GlobalClose;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control tabstrip with a global close button.

Preventing a User from Closing a Specific Tab

When tab closing enabled (see Tab Closing (page 24) and Allowing Users to Close Tabs (page 36) for more information), users can disable any tab on the strip by default. However, you can prevent users from closing a specific tab by setting that C1TabItem item's CanUserClose property to **False**.

In Design View

Complete the following steps:

1. Select the tab that you wish to prevent users from closing.
2. In the **Properties** window, clear the CanUserClose check box.

In XAML

To prevent a user from closing a tab, add `CanUserClose="False"` to the `<c1:C1TabItem>` tag of the tab you want to prevent users from closing. The XAML will resemble the following:

```
<c1:C1TabItem CanUserClose="False"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
C1TabItem1.CanUserClose = False
```

- C#

```
c1TabItem1.CanUserClose = false;
```

3. Run the program.

Adding a Menu to the Tabstrip

You can add a menu to the tabstrip so that users can open and close tabs from a context menu instead of by clicking on tabs. To add the tab menu, set the C1TabControl control's TabStripMenuVisibility property to **Visible**.

In Design View

Complete the following steps:

1. Select the C1TabControl control.
2. In the **Properties** window, click the TabStripMenuVisibility drop-down arrow and select **Visible** from the list.

In XAML

To add a menu to the tabstrip, add `TabStripMenuVisibility="Visible"` to the `<c1:C1TabControl>` tag so that the markup resembles the following:

```
<c1:C1TabControl TabStripMenuVisibility="Visible"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabControl1.TabStripMenuVisibility = Visible
     ```

   - C#

     ```
     c1TabControl1.TabStripMenuVisibility = Visible;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control tabstrip with tab menu.

Overlapping Tabs on a Tabstrip

You can control the overlapping of tabs by setting the TabStripOverlap property and the TabStripOverlapDirection property (for more information, see Tab Overlapping (page 25)). In this topic, you'll set the TabStripOverlap property and TabStripOverlapDirection properties in design view, in XAML, and in code. This topic assumes that you have added a C1TabControl with at least three tabs to your project (see Adding a Tab to the C1TabControl Control (page 31)).

In Design View

Complete the following steps:

1. Select the C1TabControl control.
2. In the Properties window, set the following properties:
   - Set the TabStripOverlap property to "12".
   - Set the TabStripOverlapDirection property to **Left**.

In XAML

Add `TabStripOverlap="12"` and `TabStripOverlapDirection="Left"` to the `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl HorizontalAlignment="Left" VerticalAlignment="Top"
Height="156" Width="226" TabStripOverlap="12"
TabStripOverlapDirection="Left">
```

In Code

Complete the following steps:

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1TabControl1.TabStripOverlap = 12

     C1TabControl1.TabStripOverlapDirection = C1.Silverlight.
     C1TabPanelOverlapDirection.Left
     ```

   - C#

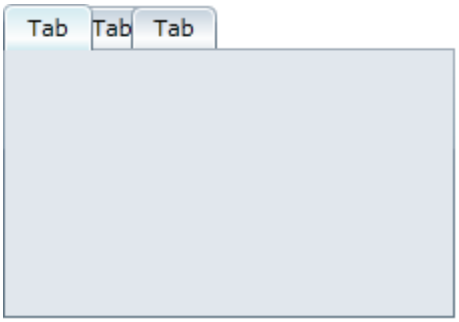
     ```
     c1TabControl1.TabStripOverlap = 12;

     C1TabControl1.TabStripOverlapDirection = C1.Silverlight.
     c1TabPanelOverlapDirection.Left;
     ```

2. Run the program.

This Topic Illustrates the Following:

The following image depicts the result of this topic.


Tab
Tab
Tab