
ComponentOne

Accordion for WPF

Copyright © 2012 ComponentOne LLC. All rights reserved.

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: <http://www.componentone.com>

Sales

E-mail: sales@componentone.com

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

Trademarks

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

Warranty

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

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

Copying and Distribution

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

This manual was produced using [ComponentOne Doc-To-Help™](#).

Table of Contents

ComponentOne Accordion for WPF	1
Installing Accordion for WPF	1
Studio for WPF Setup Files	1
Using Maps Powered by Esri	2
System Requirements	3
Installing Demonstration Versions	4
Uninstalling Accordion for WPF	4
End-User License Agreement	4
Licensing FAQs	4
What is Licensing?	4
How does Licensing Work?	5
Common Scenarios	5
Troubleshooting.....	7
Technical Support.....	9
Redistributable Files	10
About this Documentation	10
XAML and XAML Namespaces	10
Creating a Microsoft Blend Project	11
Creating a .NET Project in Visual Studio	12
Creating an XAML Browser Application (XBAP) in Visual Studio	13
Adding the Accordion for WPF Components to a Blend Project.....	14
Adding the Accordion for WPF Components to a Visual Studio Project.....	15
Key Features	17
Accordion for WPF Quick Start	17
Step 1 of 3: Creating an Application with a C1Accordion Control.....	17
Step 2 of 3: Adding Accordion Panes	18
Step 3 of 3: Running the Project.....	18
Using C1Accordion	20
C1Accordion Elements	21
Accordion Pane Header.....	21

Accordion Pane Content Area	22
Expanding and Collapsing Accordion Panes	22
Accordion for WPF Layout and Appearance	25
ComponentOne ClearStyle Technology	26
How ClearStyle Works	26
C1Accordion and C1AccordionItem ClearStyle Properties.....	26
Accordion for WPF Appearance Properties.....	27
Text Properties.....	27
Content Positioning Properties.....	28
Color Properties.....	28
Border Properties.....	29
Size Properties	29
Templates	29
C1Accordion Themes.....	30
Accordion for WPF Samples.....	33
Accordion for WPF Task-Based Help.....	35
Adding Accordion Panes to the C1Accordion Control	35
Adding Content to Header Elements	36
Adding Text to the Header	36
Adding a Control to the Header.....	36
Adding Content to Content Areas.....	37
Adding Text to the Content Area.....	38
Adding a Control to the Content Area	38
Adding Multiple Controls to the Content Area	40
Changing the Expand Direction	40
Filling Out the Accordion's Height	41

ComponentOne Accordion for WPF

Display a list of expandable items with **ComponentOne Accordion™ for WPF**. Select an item to expand it and collapse all others, automatically organizing your UI and optimizing the use of screen real estate.

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



Getting Started

- [Using C1Accordion](#) (page 20)
- [Quick Start](#) (page 17)
- [Task-Based Help](#) (page 35)

Installing Accordion for WPF

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

Studio 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 **ComponentOne Accordion 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
- C1.WPF.Extended.dll
- C1.WPF.Extended.Expression.Design.dll
- C1.WPF.Extended.Expression.Design.4.0.dll
- C1.WPF.Extended.VisualStudio.Design.dll
- C1.WPF.Extended.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 C1Accordion 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 Viewer help to the C:\Program Files\ComponentOne\Studio for WPF directory in the following folder:

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\\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\\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 **Accordion 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**.

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\\<version number>

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

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

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

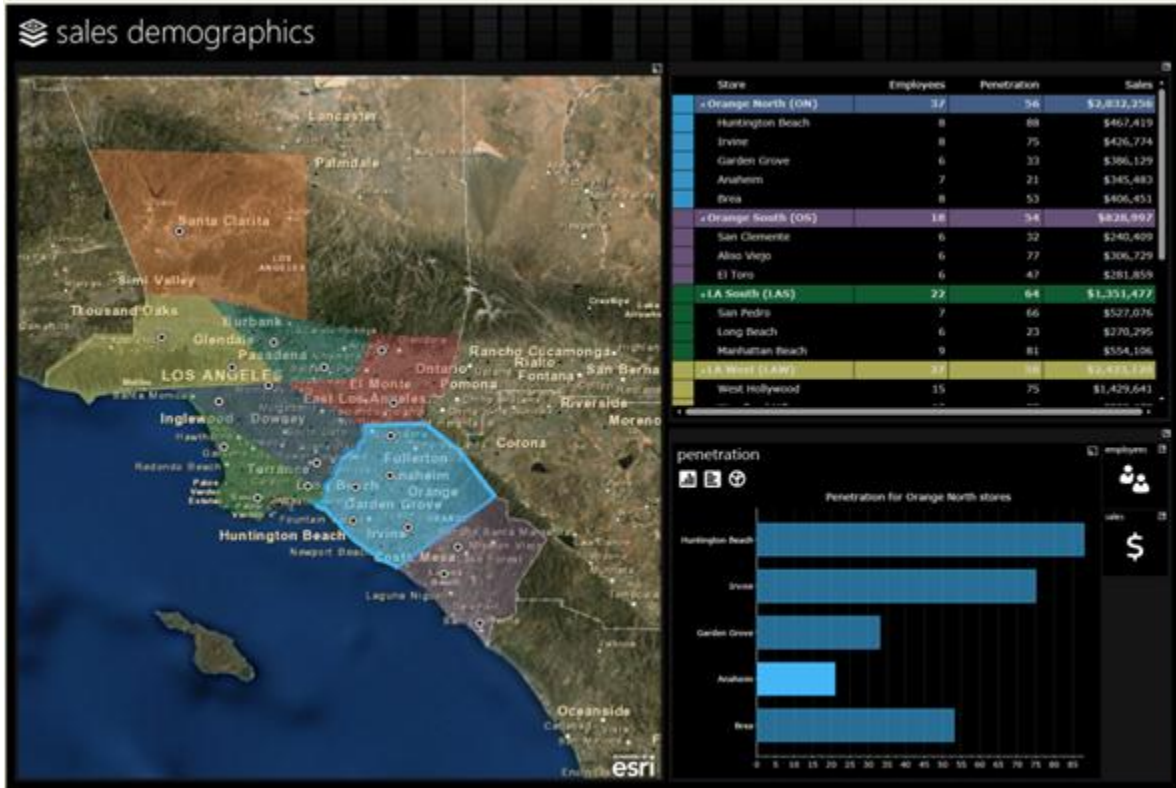
Using Maps Powered by Esri

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

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

Benefits of Maps powered by Esri:

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



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

System Requirements

System requirements include the following:

Operating Systems:

Microsoft Windows® XP with Service Pack 2 (SP2)
 Windows Vista™
 Windows 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®

Accordion for WPF includes design-time support for Expression

Blend Compatibility: 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 **Accordion 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 Accordion 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 Accordion for WPF

To uninstall **ComponentOne Studio for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs (Programs and Features in 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: Cl.Win.ClFlexGrid.ClFlexGrid  
{  
    // ...  
}
```

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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

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

Using licensed components with automated testing products

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

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

For example:

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

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

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

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

Troubleshooting

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

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

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

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

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

If that fails follow these steps:

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

Alternatively, follow these steps:

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

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

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

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **[Online Resources](#)**
ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, [samples and videos](#), Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
This online support service provides you with direct access to our Technical Support staff via an [online incident submission form](#). When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums**
ComponentOne peer-to-peer product [forums](#) 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 Accordion for WPF is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.dll
- C1.WPF.Extended.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

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.

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

ComponentOne

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

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

412.681.4343

412.681.4384 (Fax)

<http://www.componentone.com/>

ComponentOne Doc-To-Help

This documentation was produced using [ComponentOne Doc-To-Help® Enterprise](#).

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
<code>xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"</code>	This is the default Windows Presentation Foundation namespace.
<code>xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"</code>	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 `ClAccordion` 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.

The namespace value is `my` and the namespace is `C1.WPF.Extended`.

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

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

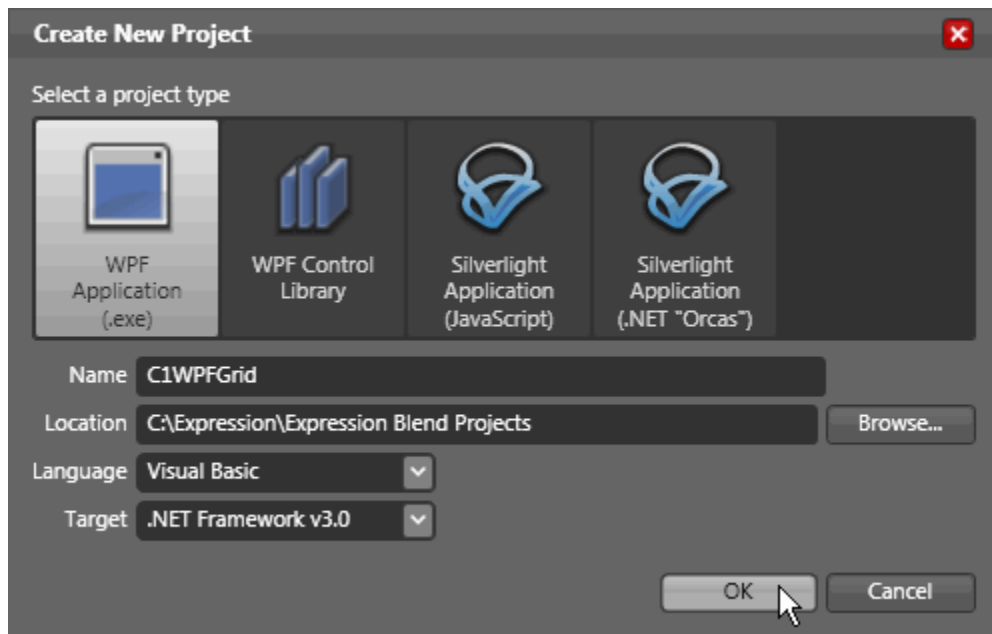
You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the panel:

```
<MyMTB:ClAccordion Name="clAccordion1" 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**.

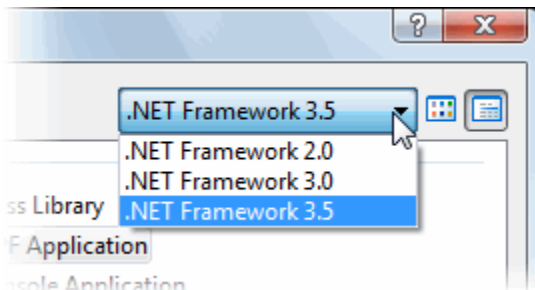


A new Blend project with a XAML window is created.

Creating a .NET Project in Visual Studio

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

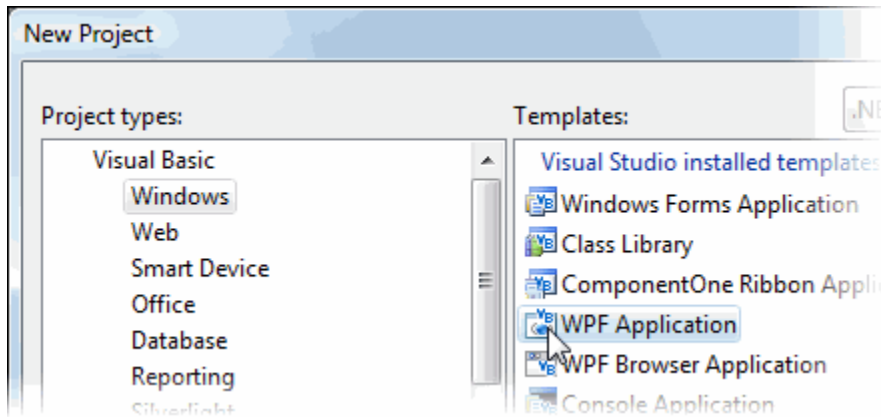
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.
The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



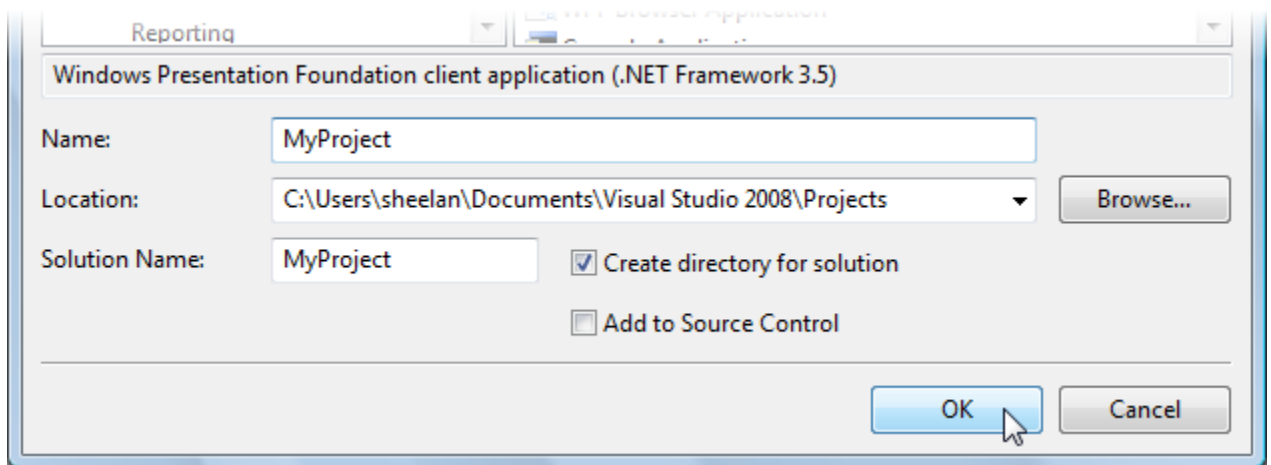
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

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**.



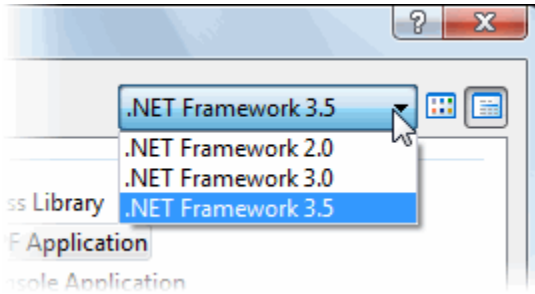
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

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 Accordion for WPF Components to a Blend Project

In order to use **C1Accordion** or another **ComponentOne Accordion 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.Extended.dll** assembly installed with **Accordion for WPF**.

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

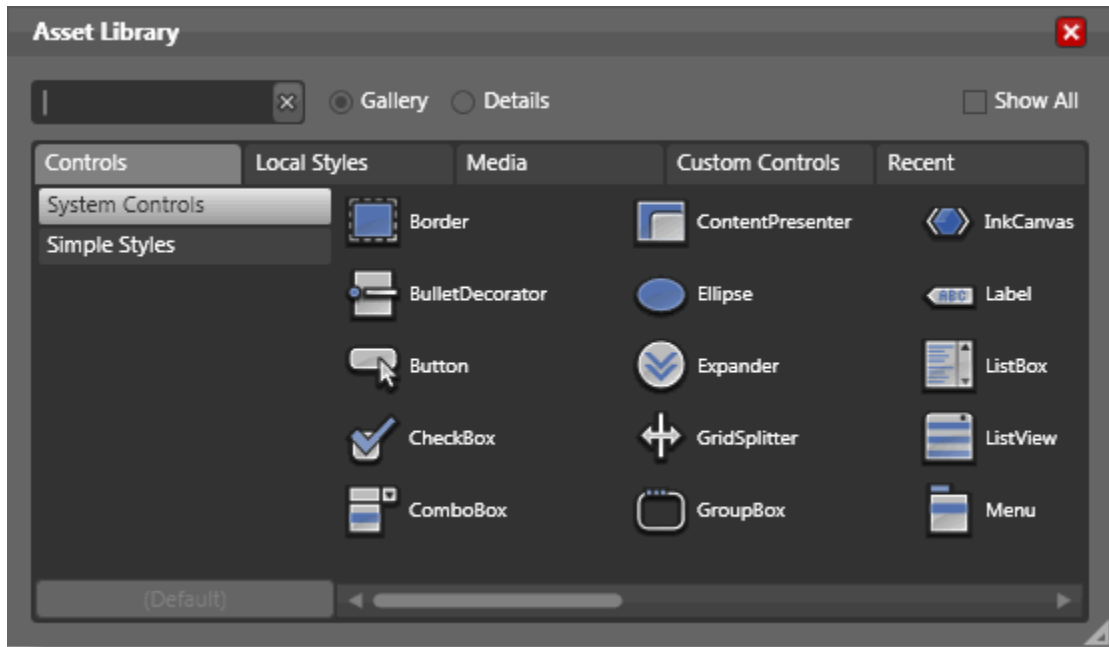
3. Select **C1.WPF.Extended.dll** and click **Open**. A reference is added to your project.

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 **Accordion for WPF** main and auxiliary components are listed here.
3. Select **C1Accordion**. The component will appear in the Toolbox above the **Asset Library** button.
4. Double-click the **C1Accordion** component in the Toolbox to add it to **Window1.xaml**.

Adding the Accordion for WPF Components to a Visual Studio Project

When you install **ComponentOne Accordion for WPF** the C1Accordion control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne Accordion for WPF provides the following control:

- C1Accordion

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

Manually Adding Accordion for WPF to the Toolbox

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

- C1Accordion

To manually add the C1Accordion 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 **Accordion for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFAccordion**, 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.

- 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 Accordion for WPF to the Window

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

- Add the C1Accordion control to the Visual Studio Toolbox.
- Double-click C1Accordion or drag the control onto the window.

Adding a Reference to the Assembly

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

- Select the **Add Reference** option from the **Project** menu of your project.
- Select the **ComponentOne Accordion 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**.
- 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 **Accordion for WPF** assembly visible to the project.

Key Features

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

- **Expand Direction**

The **C1Accordion** control has the ability to expand in four different directions. The **ExpandDirection** property indicates which direction the control expands and can be set to **Top**, **Right**, **Bottom**, or **Left**. For more information, see the **ExpandDirection** topic.

- **Custom Header**

An accordion pane's header can be customized with both text and controls. For more information on the customizable header element, see [Accordion Pane Header](#) (page 21).

- **Configure Items in an Organized Pattern**

Accordion is designed to maximize space. Configure the size and position of **C1Accordion** to hide items until needed.

- **Add Objects of any Data Type**

Because the **C1Accordion** control inherits from **ItemsControl**, you can add objects of any data type to its **Items** collection and use a **DataTemplate** to create a visual representation of the items.

Accordion for WPF Quick Start

The following quick start guide is intended to get you up and running with **Accordion for WPF**. In this quick start, you'll start in Visual Studio to create a new project with a **C1Accordion** control. You will also customize the accordion, add accordion panes filled with content to it, and then observe some of the run-time features of the control.

Step 1 of 3: Creating an Application with a C1Accordion Control

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

Complete the following steps:

1. In Visual Studio 2008, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **WPF Application**.
3. Enter a **Name** for your project and click **OK**.
4. Navigate to the Toolbox and double-click the **C1Accordion** icon to add the control to the project.
5. Right click the **C1Accordion** control to open its context menu and then select **Properties**.

The **Properties** window opens with the **C1Accordion** control's properties in focus.

6. Set the following properties.
 - Set the **Height** property to "250" to set the height of the control.
 - Set the **Width** property to "400" to set the width of the control.
 - Set the **ExpandDirection** property to **Left** so that the **C1Accordion** control will expand from the bottom rather than expanding from the top, which is its default.

- Set the **Fill** property to **True** by selecting the **Fill** check box. This means that each pane will expand to fill the specified width of the **C1Accordion** control.
- Set **AllowCollapseAll** to **False** by clearing the **AllowCollapseAll** check box. This will prevent users from collapsing all panes at the same time.

You've successfully created a WPF application containing a **C1Accordion** control. In the next step, you will customize the appearance and behavior of the **C1Accordion** control.

Step 2 of 3: Adding Accordion Panes

In the last step, you created a project with a customized **C1Accordion** control. In the next step, you will add accordion panes, which you will customize and add content to.

Complete the following steps:

1. Click the **C1Accordion** control once to select it.
2. In the **Properties** window, click the **Items** ellipsis button.
The **Collection Editor: Items** dialog box opens.
3. Click the **Add** button three times to add three **C1AccordionItem** items to the **C1Accordion** control.
4. Select the first **C1AccordionItem** and set the following properties:
 - Set the **Background** property to **Aqua** to set the background color of the accordion pane.
 - Set the **Content** property to "This is text content" to add text content to the accordion pane.
5. Select the second **C1AccordionItem** and set the following properties:
 - Set the **Background** property to **AliceBlue** to set the background color of the accordion panel.
 - Set the **IsExpanded** property to **True** so that this pane will be expanded at run time.
6. Select the third **C1AccordionItem** and set the **Background** property to **LawnGreen** to set the background color of the accordion pane.
7. Click **OK** to close the **Collection Editor: Items** dialog box.
8. Switch to XAML view and complete the following:
 - Add `Header="Pane 1"` to the first `<c1:C1AccordionItem>` tag.
 - Add `Header="Pane 2"` to the second `<c1:C1AccordionItem>` tag.
 - Add `Header="Pane 3"` to the third `<c1:C1AccordionItem>` tag.
9. Switch to Design view and add a control to the second accordion pane by completing the following steps:
 - a. In the Designer, click the second accordion pane to select it.
 - b. Navigate to the Toolbox and double-click the **Calendar** icon to add the control to the accordion pane.

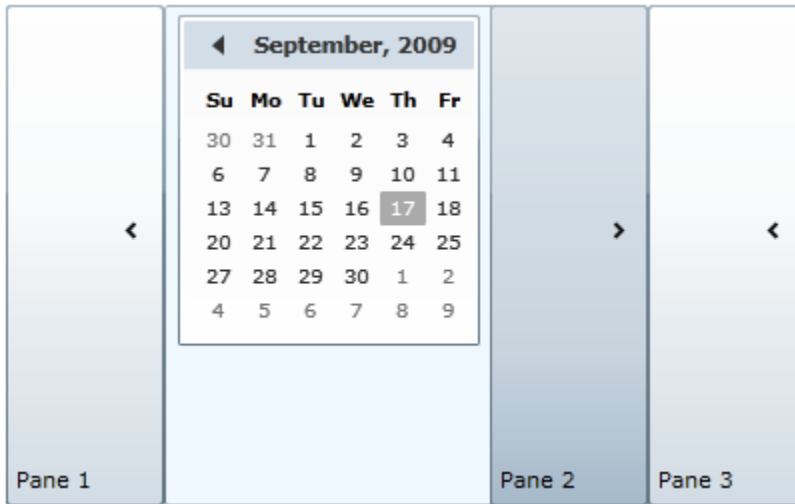
In this step, you added three accordion panes to the **C1Accordion** control and then added content to two of the accordion panes. In the next step, you will run the project and observe the run-time features of the control.

Step 3 of 3: Running the Project

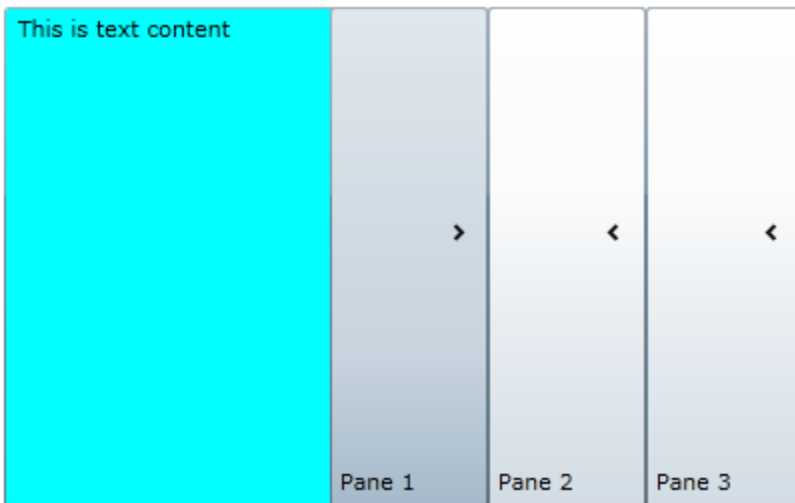
In the last step, you added accordion panes and content to the **C1Accordion** control. In this step, you will run the project and observe some of the run-time features of the **C1Accordion** control.

Complete the following steps:

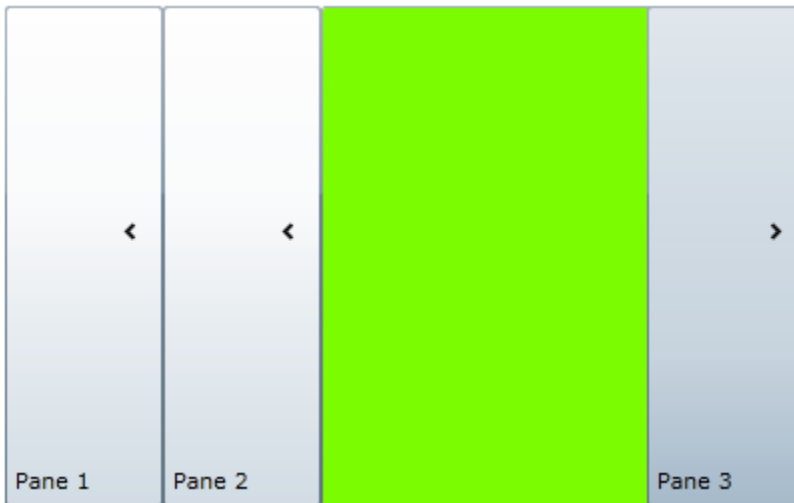
1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe that the second pane, which holds the **Calendar** control, is expanded.



2. Click **Pane 1** and observe that the first pane expands to reveal its content.



3. Click **Pane 3** and observe that the last pane expands. Note that the third pane expands to the same width as the other panes despite the fact that it has no content.



4. Click **Pane 3** and observe that you can't close the pane. This is because you set the `AllowCollapseAll` property to **False**, which means that one accordion pane must be expanded at all times.

Congratulations! You have completed the **Accordion for WPF** quick start tutorial. In this tutorial, you created a WPF project containing a `C1Accordion` control, modified the appearance and behavior of the control, added accordion panes and accordion pane content to the control, and then observed some of the run-time features of the control.

Using C1Accordion

The `C1Accordion` control is a container that can hold a series of expandable and collapsible panes for storing text, images, and controls. The `C1Accordion` control is an **ItemsControl**, which means that the control is designed to host a series of objects. The `C1AccordionItem` class represents the items, or accordion panes, that can be hosted by the `C1Accordion` control.

When you add the `C1Accordion` 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 multiple accordion panes to it in Design view, in XAML, or in code. The following image depicts a `C1Accordion` control with three accordion pane items, the first of which is expanded.



The image above shows the accordion panes sans header text or content, but customizing the header and adding content to the pane is as simple as setting a few properties. You can also modify behaviors, such as the expandability and the direction, of the control.

The following topics provide an overview of the `C1Accordion` control's elements and features.

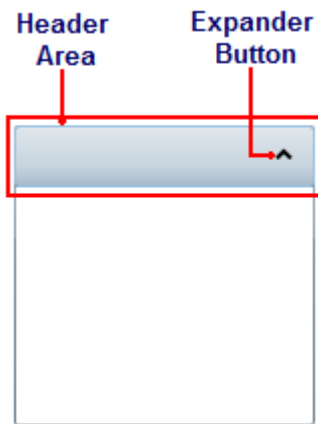
C1Accordion Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Accordion control. The control is comprised of two elements – the header and the content area – which combine to make the complete C1Accordion control.

Accordion Pane Header

By default, the header element of an accordion pane appears at the top of the control and the expander button appears on the right side of the header. When the C1AccordionItem item (accordion pane) is first placed on the page, the header element contains no text.

The following image labels the header area of an accordion pane.



To add text to the header element, simply set the **Header** property to a string. Once the text is added, you can style it using several font properties (see [Text Properties](#) (page 27)). You can also add WPF controls to the header.

The placement of the header element and expander button will change depending on the expand direction of the control. For more information on expand directions, see the [Expand Direction](#) (page 23) topic.

Attribute Syntax versus Property Element Syntax

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

```
<c1ext:C1AccordionItem Header="Hello World"/>
```

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

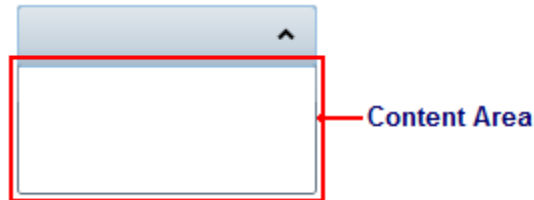
```
<c1ext:C1AccordionItem Width="150" Height="55" Name="C1AccordionItem1">
  <c1ext:C1AccordionItem.Header>
    <Grid HorizontalAlignment="Stretch">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="C1AccordionItem" />
    </Grid>
  </c1ext:C1AccordionItem.Header>
</c1ext:C1AccordionItem>
```

```
</c1ext:C1AccordionItem.Header>
</c1ext:C1AccordionItem>
```

Accordion Pane Content Area

An accordion pane's content area initially consists of an empty space. In the content area, you can add grids, text, images, and arbitrary controls. When working in Blend or Visual Studio's Design view, elements in the content area of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the content area of an accordion pane.



You can add text to the content area by setting the item's **Content** property or by adding a **TextBox** element to the content area. Adding WPF elements to the content area 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 `C1AccordionItem` 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 `C1AccordionItem` item, but its ability to hold multiple elements will allow you to show several controls in the content area of the accordion pane.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the content area, 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:

```
<c1ext:C1AccordionItem Content="Hello World"/>
```

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

```
<c1ext:C1AccordionItem Width="150" Height="55" Name="C1AccordionItem1">
  <c1ext:C1AccordionItem.Content>
    <StackPanel>
      <TextBlock Text="Hello"/>
      <TextBlock Text="World"/>
    </StackPanel>
  </c1ext:C1AccordionItem.Content>
</c1ext:C1AccordionItem>
```

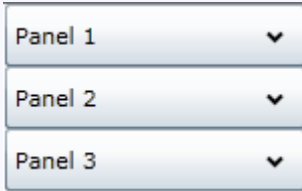
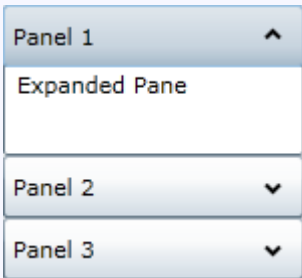
Expanding and Collapsing Accordion Panes

This section details the options for customizing the way that the accordion panes expand and collapse.

Accordion Pane Initial Expand State

By default, the **IsExpanded** property of each accordion pane is set to **False**, which means that the pane appears in its collapsed state when the page is loaded. If you want the pane to be expanded upon page load, you can set the **IsExpanded** property to **True**.

The following table illustrates the difference between the two expand states.

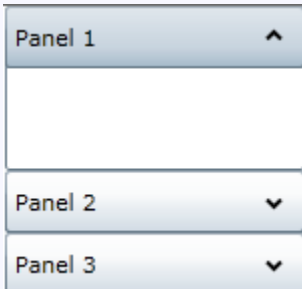
IsExpanded	Result
IsExpanded=False	
IsExpanded=True	

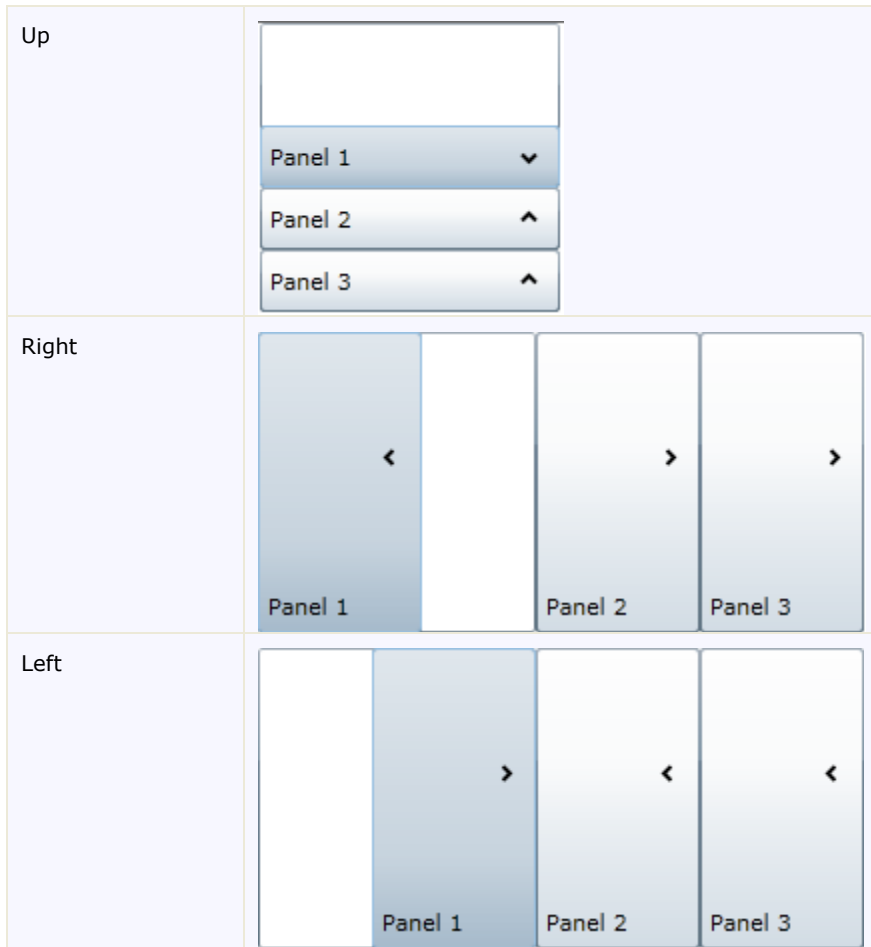
You can set the expand states in Design view, in XAML, or in code.

Expand Direction

The C1Accordion control includes the option to specify the expand direction using the **ExpandDirection** property. In addition to setting the direction that the control expands, changing the **ExpandDirection** also changes the header's orientation to the content area of the control. By default the **ExpandDirection** property is set to **Down** and the control expands from top to bottom.

The following table illustrates each **ExpandDirection** setting.

ExpandDirection	Result
Down	



You can set the collapsing and expanding direction in Design view, in XAML, or in code.

Collapsing Panes

By default, the `C1Accordion` control's `AllowCollapseAll` property is set to **True**, meaning that all panes of an accordion can be collapsed by the user. This is useful in cases where you want to conserve screen real estate so that the user is free of unnecessary distractions. Think of it like the musical instrument accordion, which can be tightly compressed for storage and then decompressed when a player wants to use it.

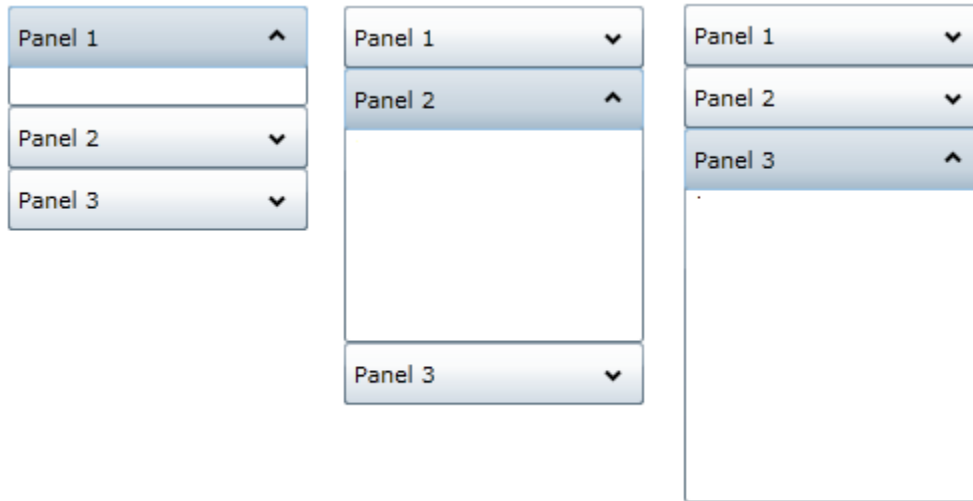
But there are times when this won't fit into the design of your user interface. For example, you may want to use the accordion as a menu element. In that case, you'd want the `C1Accordion` control to fill to a specific height or width at all times, and the control won't fulfill that need if all of its panes can be collapsed. Therefore, you would set the `AllowCollapseAll` property to **False** so that one pane has to remain open at all times. You would then set the height or width property as desired and then set the `Fill` property to **True** so that the `C1Accordion` control and its panes always fill to your specified height or width.

Note: The `C1Accordion` control is only capable of expanding one pane at a time.

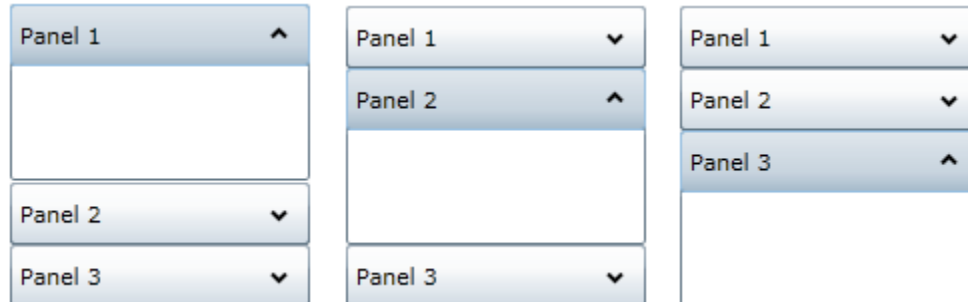
Expansion Fill

The default behavior of an accordion pane (`C1AccordionItem`) is to fill only to the height (for accordions that expand up or down) or width (for accordions that expand right or left) of an accordion pane's contents. This will

cause the accordion to grow to different heights or widths depending on the height or width of the content for a given pane. This is illustrated in the example below.



To avoid this, set the height or width of the C1Accordion control and then set its Fill property to **True**. This will cause each panel to expand to fill the width or height that you specified, thus providing a uniform width or height for accordion pane. You can observe the difference in the example below.



Accordion for WPF Layout and Appearance

The following topics detail how to customize the C1Accordion 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.

C1Accordion and C1AccordionItem ClearStyle Properties

Accordion 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 **C1Accordion** control:

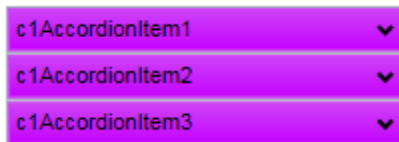
Brush	Description
Background	Gets or sets the brush of the control's background.
ExpandedBackground	Gets or sets the brush of the header background that appears when items of the C1Accordion control are expanded.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control's C1AccordionItems when it is moused over.
HeaderBackground	Gets or sets the background brush of all headers of all C1AccordionItem headers contained within the C1Accordion control.

The following table outlines the brush properties of the **C1AccordionItem** control:

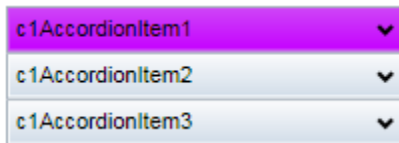
Brush	Description
-------	-------------

Background	Gets or sets the brush of the control's background.
ExpandedBackground	Gets or sets the brush of the header background when the C1AccordionItem is expanded.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control when it is moused over.
HeaderBackground	Gets or sets the background brush of the C1AccordionItem header.

You can completely change the appearance of the **C1Accordion** and **C1AccordionItem** controls by setting a few properties, such as the **C1Accordion** control's **HeaderBackground** property, which sets the background color for every accordion item in a **C1Accordion** control. For example, if you set the **C1Accordion** control's **HeaderBackground** property to "#FFC500FF", each header in the **C1Accordion** control would appear similar to the following:



You can also choose to modify the accordion items independently by setting a **C1AccordionItem** control's **HeaderBackground** property. For example, if you set a the **HeaderBackground** property of the first accordion item to "#FFC500FF", the control would appear similar to the following:



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

Accordion for WPF Appearance Properties

ComponentOne Accordion 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 accordion pane.

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 accordion pane content area. This is a dependency property.
Header	Gets or sets the header of an accordion 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 C1Accordion control.

Property	Description
HeaderPadding	Gets or sets the padding of the header.
HeaderHorizontalContentAlignment	HorizontalAlignment 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 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.
HeaderBackground	
HeaderForeground	Gets or sets the foreground brush of the

	header.
--	---------

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 **C1Accordion** control.

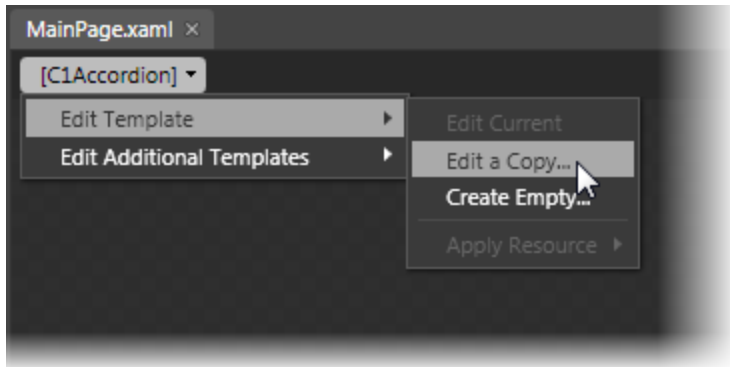
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne Accordion 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 C1Accordion 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 C1AccordionItem template, simply select the C1AccordionItem 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.

C1Accordion Themes

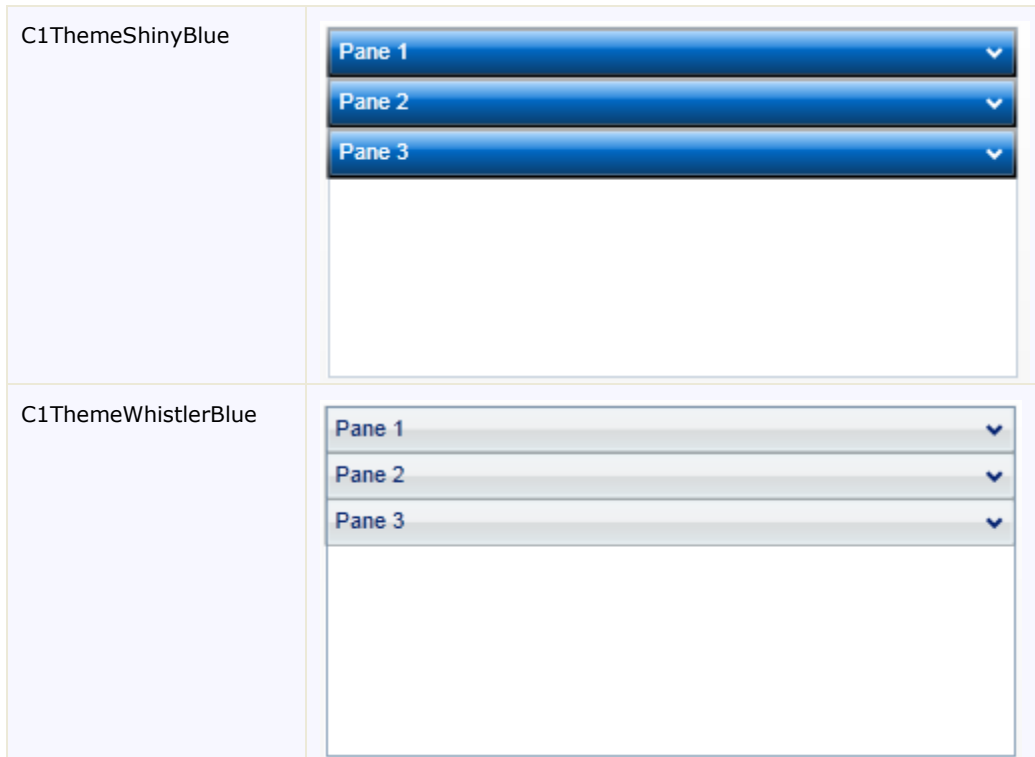
ComponentOne Accordion for WPF incorporates several themes that allow you to customize the appearance of your grid. When you first add a C1Accordion control to the page, it appears similar to the following image:



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

Theme Name	Theme Preview
------------	---------------

C1ThemeBureauBlack	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 1 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 2 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 3 ▼</div> <div style="height: 100px;"></div> </div>
C1ThemeExpressionDark	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 1 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 2 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 3 ▼</div> <div style="height: 100px;"></div> </div>
C1ThemeExpressionLight	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 1 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 2 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 3 ▼</div> <div style="height: 100px;"></div> </div>
C1Blue	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 1 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 2 ▼</div> <div style="border-bottom: 1px solid #ccc; padding: 2px 5px;">Pane 3 ▼</div> <div style="height: 100px;"></div> </div>



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

- Visual Basic

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

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

- C#

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

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

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

- Visual Basic

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

    ' Using Merged Dictionaries
```

```
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

- C#

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

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

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

Accordion 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 C1Accordion control:

Sample	Description
Accordion	Illustrates the functionality of the C1Accordion control.

Accordion 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 C1Accordion control in general. If you are unfamiliar with the **ComponentOne Accordion for WPF** product, please see the **Accordion for WPF Quick Start** first.

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

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

Adding Accordion Panes to the C1Accordion Control

In this topic, you will add an accordion pane to a C1Accordion control in Design view, in XAML, and in code.

At Design Time in Design view

To add a pane to the C1Accordion control, complete the following steps:

1. Click the C1Accordion control once to select it.
2. In the **Properties** window, click the **Items** ellipsis button.
3. The Collection Editor: Items dialog box opens.
4. Click the **Add** button once to add one C1AccordionItem item to the C1Accordion control.
5. In the **Properties** grid, set the **Width** property to "150".

In XAML

To add a pane to the C1Accordion control, place the following markup between the `<c1ext:C1Accordion>` and `</c1ext:C1Accordion>` tags:

```
<c1ext:C1AccordionItem Name="C1AccordionItem" Width="150">
</c1ext:C1AccordionItem>
```

In Code

To add accordion panes in code, complete the following steps:

1. Enter Code view and import the following namespace:
 - Visual Basic
`Imports C1.WPF.Extended`
 - C#
`using C1.WPF.Extended;`
2. Add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
'Create the accordion pane and add content
Dim C1AccordionItem1 As New C1AccordionItem()
C1AccordionItem1.Content = "C1AccordionItem1"
'Add the accordion pane to the C1Accordion control
C1Accordion1.Items.Add(C1AccordionItem1)
```
- C#

```
//Create the accordion pane and add content
```

```
C1AccordionItem c1AccordionItem1 = new C1AccordionItem();
C1AccordionItem1.Content = "c1AccordionItem1";
//Add the accordion pane to the C1Accordion control
c1Accordion1.Items.Add(c1AccordionItem1);
```

3. Run the program.

Adding Content to Header Elements

You can easily add both simple text and WPF controls to an accordion pane's header. The topics in this section will provide step-by-step instructions about adding text content and controls to the header.

For more information on the header element, you can also visit the [Accordion Pane Header](#) (page 21) topic.

Adding Text to the Header

By default, an accordion pane's header is empty. You can add text to the control's header by setting the **Header** property to a string in Design view, in XAML, or in code.

This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

In XAML

To set the **Header** property in XAML, add `Header="Hello World"` to the `<c1ext:C1AccordionItem>` tag so that it appears similar to the following:

```
<c1ext:C1AccordionItem Name="C1AccordionItem1" Header="Hello World"
Width="150" Height="55">
```

In Code

To set the **Header** property in code, complete the following steps:

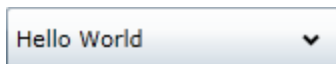
1. Enter Code view and add the following code beneath the **InitializeComponent()** method:
 - Visual Basic

```
C1AccordionItem1.Header = "Hello World"
```
 - C#

```
c1AccordionItem1.Header = "Hello World";
```
2. Run the program.

✔ This Topic Illustrates the Following:

The header of the accordion pane now reads "Hello World". The result of this topic should resemble the following:



Adding a Control to the Header

Any accordion pane header element is able to accept a WPF control. In this topic, you will add a **Button** control to the header in XAML and in code.

This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

In XAML

To add a **Button** control to the header in XAML, place the following XAML markup between the `<clext:C1AccordionItem>` and `</clext:C1AccordionItem>` tags:

```
<clext:C1AccordionItem.Header>
  <Button Content="Button" Height="Auto" Width="50"/>
</clext:C1AccordionItem.Header>
```

In Code

To add a **Button** control to the header in code, complete the following steps:

1. Enter Code view and 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 = 50
NewButton.Height = Double.NaN
'Add the Button to the header
C1AccordionItem1.Header = (NewButton)
```

- C#

```
InitializeComponent();
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = 50;
NewButton.Height = Double.NaN;
//Add the Button to the header
c1AccordionItem1.Header = (NewButton);
```

2. Run the program.

✔ This Topic Illustrates the Following:

As a result of this topic, the control will appear in the header. The final result will resemble the following image:



Adding Content to Content Areas

You can easily add both simple text and WPF controls to an accordion pane's header. The topics in this section will provide step-by-step instructions about how to add text content and controls to the header.

For more information on the header element, you can also visit the [Accordion pane Content Area](#) (page 22) topic.

Adding Text to the Content Area

You can easily add a simple line of text to the content area of an accordion pane by setting the **Content** property to a string in Design view, in XAML, or in code.

This topic assumes that you have added a `C1Accordion` control with at least one `C1AccordionItem` item to your project.

Note: You can also add text to the content area by adding a **TextBox** control to the content area and then setting the **TextBox** control's **Text** property. To learn how to add a control to the content area, see [Adding a Control to the Content Area](#) (page 38).

At Design Time in Design view

To set the **Content** property in Design view, complete the following steps:

1. Click the `C1AccordionItem` item once to select it.
2. In the **Properties** window, set the **Content** property to a string (for example, "Hello World").
3. Run the program and expand the accordion pane.

In XAML

To set the **Content** property in XAML, complete the following:

1. Add `Content="Hello World"` to the `<c1ext:C1AccordionItem>` tag so that it appears similar to the following:

```
<c1ext:C1AccordionItem Name="C1AccordionItem1" Content="Hello World" Width="150" Height="55">
```

2. Run the program and expand the accordion pane.

In Code

To set the **Content** property in code, complete the following steps:

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

- Visual Basic

```
C1AccordionItem1.Content = "Hello World"
```

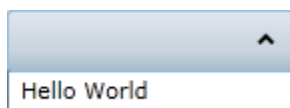
- C#

```
c1AccordionItem1.Content = "Hello World";
```

2. Run the program and expand the accordion pane.

✔ This Topic Illustrates the Following:

The content of your accordion pane now reads "Hello World". The result of this topic should resemble the following:



Adding a Control to the Content Area

Each accordion pane (`C1AccordionItem`) will accept one child control in its content area. In this topic, you will add a WPF button control in Design view, in XAML, and in code.

This topic assumes that you have added a `C1Accordion` control with at least one `C1AccordionItem` item to your project.

At Design Time in Design view

To add a **Button** control to the content area, complete the following steps:

1. Select the accordion pane you wish to add the control to.
2. Double click the **Button** icon to add it to the accordion pane's content area.
3. In the designer, select the **Button** control so that its properties take focus in the **Properties** window.
4. Set the **Width** property to "Auto".
5. Set the **Height** property to "Auto".
6. Run the program and expand the accordion pane to reveal the button control.

In XAML

Complete the following steps:

1. To add a **Button** control to the content area in XAML, complete the following:
2. Place the following markup between the `<c1ext:C1AccordionItem>` and `</c1ext:C1AccordionItem>` tags:

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

3. Run the program and expand the accordion pane to reveal the button control.

In Code

To add a **Button** control to the content area in code, complete the following:

1. Enter Code view and 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
C1AccordionItem1.Content = (NewButton)
```

- C#

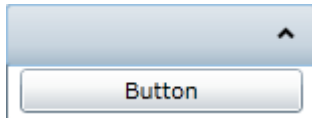
```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
```

```
//Add the Button to the content area
c1AccordionItem1.Content = (NewButton);
```

2. Run the program and expand the accordion pane to reveal the button control.

✔ This Topic Illustrates the Following:

When accordion pane is expanded, the button control will appear in its content area, resembling the following image:



Adding Multiple Controls to the Content Area

You cannot set an accordion pane's (C1AccordionItem) **Content** property to more than one control at a time. However, you can circumvent this issue by adding a panel-based control that can accept more than one control, such as a **StackPanel** control, to the content area of the accordion pane. When you add multiple controls to the panel-based control, each one will appear within the accordion pane's content area.

This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

To add multiple controls to the content area, complete these steps:

1. Place the following XAML markup between the `<c1ext:C1AccordionItem>` and `</c1ext:C1AccordionItem>` tags:

```
<c1ext:C1AccordionItem.Content>
  <StackPanel>
    <TextBlock Text="1st TextBlock"/>
    <TextBlock Text="2nd TextBlock"/>
    <TextBlock Text="3rd TextBlock"/>
  </StackPanel>
</c1ext:C1AccordionItem.Content>
```

2. Run the program.
3. Expand the accordion pane and observe that each of the three **TextBlock** controls appear in the content area. The result will resemble the following:



Changing the Expand Direction

By default, the C1Accordion control's accordion panes expand from top-to-bottom because the `ExpandDirection` property is set to **Down**. You can easily change the expand direction by setting the `ExpandDirection` property to **Up**, **Right**, or **Left** in Design view, in XAML, or in code.

This topic assumes that you have added a `C1Accordion` control with at least one `C1AccordionItem` to your project.

At Design Time in Design view

To set the `ExpandDirection` property in Design view, complete the following steps:

1. Click the `C1Accordion` control once to select it.
2. In the **Properties** window, click the `ExpandDirection` drop-down arrow and select one of the options from the list. For this example, select **Right**.

In XAML

To set the `ExpandDirection` property to **Right** in XAML, add `ExpandDirection="Right"` to the `<clext:C1Accordion>` tag so that it appears similar to the following:

```
<clext:C1Accordion Name="C1Accordion1" ExpandDirection=Right  
Width="150" Height="55">
```

In Code

To set the `ExpandDirection` property in code, complete the following steps:

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1Accordion1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right`
 - C#
`C1Accordion1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right;`
2. Run the program.

✔ This Topic Illustrates the Following:

By following the instructions in this topic, you have learned how to set the `ExpandDirection` property. In this topic, you set the `ExpandDirection` property to **Right**, which will make the `C1Accordion` control resemble the following:



Filling Out the Accordion's Height

The default behavior of an accordion pane (`C1AccordionItem`) is to fill only to the height (for accordions that expand up or down) or width (for accordions that expand right or left) of its contents. If you want the `C1Accordion` control to fill to a specific height or width, you will have to set the height or width of the control and then set the control's `Fill` property to **True**.

At Design Time in Design view

To set the `Fill` property in Design view, complete the following steps:

1. Add three accordion panes to your `C1Accordion` control (see [Adding Accordion Panes to the C1Accordion Control](#) (page 35)).
2. Click the `C1Accordion` control once to select it.
3. In the **Properties** window, complete the following:

- Select the **Fill** check box.
- Set the **Height** property to "200".

In XAML

To set the Fill property to **True** in XAML, complete the following:

1. Add `Fill="True"` to the `<c1ext:C1Accordion>` tag so that it appears similar to the following:

```
<c1ext:C1Accordion Name="C1Accordion1" Fill="True">
```

2. Add `Height="200"` to the `<c1ext:C1Accordion>` tag so that it appears similar to the following:

```
<c1ext:C1Accordion Name="C1Accordion1" Fill="True" Height="200">
```

In Code

To set the Fill property in code, complete the following steps:

1. Add and `Height="200"` to the `<c1ext:C1Accordion>` tag. This ensures that each accordion pane expands to fill a height of 200 pixels.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1Accordion1.Fill = True`
 - C#
`c1Accordion1.Fill = true;`
3. Run the program.