

---

ComponentOne

# HyperPanel for WPF

Copyright © 2012 ComponentOne LLC. All rights reserved.

*Corporate Headquarters*

**ComponentOne LLC**

201 South Highland Avenue  
3<sup>rd</sup> Floor  
Pittsburgh, PA 15206 • USA

**Internet:** [info@ComponentOne.com](mailto:info@ComponentOne.com)

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

**Sales**

E-mail: [sales@componentone.com](mailto: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 HyperPanel for WPF Overview .....	1
Installing HyperPanel for WPF .....	1
Studio for WPF Setup Files.....	1
Using Maps Powered by Esri .....	2
System Requirements .....	3
Installing Demonstration Versions.....	4
Uninstalling HyperPanel 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 HyperPanel for WPF Components to a Blend Project .....	14
Adding the HyperPanel for WPF Components to a Visual Studio Project .....	14
Key Features .....	15
HyperPanel for WPF Quick Start .....	17
Step 1 of 3: Setting up the Application.....	17
Step 2 of 3: Adding Content to the Panel.....	18
Step 3 of 3: Customizing the Application .....	19
Working with HyperPanel for WPF.....	23
Zoom Center .....	23
Edge Opacity.....	24

Horizontal and Vertical Orientation .....	25
Flow Direction .....	26
Alignment.....	27
Content Alignment.....	28
Distribution .....	28
Scale .....	29
HyperPanel for WPF Samples .....	31
HyperPanel for WPF Task-Based Help .....	31
Adding a Control to the Panel .....	31
Changing the Background Color.....	32
Setting the Start Position.....	33
Changing the Scale.....	34
Setting the Orientation.....	35
Setting Element Distribution.....	36

# ComponentOne HyperPanel for WPF Overview

**ComponentOne HyperPanel™ for WPF** is a **StackPanel** that provides an automatic zoom effect for items near the mouse. You can place any elements in the panel to achieve carousel-like effects and display a large number of elements in a relatively small container, without using scrollbars. The resulting effect is similar to the system toolbar (dock) seen in Mac OS X.

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



## Getting Started

Get started with the following topics:

- [Key Features](#) (page 15)
- [Quick Start](#) (page 17)
- [Task-Based Help](#) (page 31)

## Installing HyperPanel for WPF

The following sections provide helpful information on installing **ComponentOne HyperPanel 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 **Component HyperPanel for WPF**, the following DLLs are installed:
- C1.WPF.dll
  - C1.WPF.Expression.Design.dll
  - C1.WPF.VisualStudio.Design.dll
  - C1.WPF.Expression.Design.4.dll
  - C1.WPF.VisualStudio.Design.4.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 C1HyperPanel 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:

<b>H2Help</b>	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
<b>HelpViewer</b>	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 Vista machines:

**Windows XP path:** C:\Documents and Settings\\My Documents\ComponentOne Samples

**Windows Vista path:** C:\Users\\Documents\ComponentOne Samples

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

<b>Common</b>	Contains support and data files that are used by many of the demo programs.
<b>Studio WPF</b>	Contains <b>Studio for WPF</b> samples.

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

## 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.

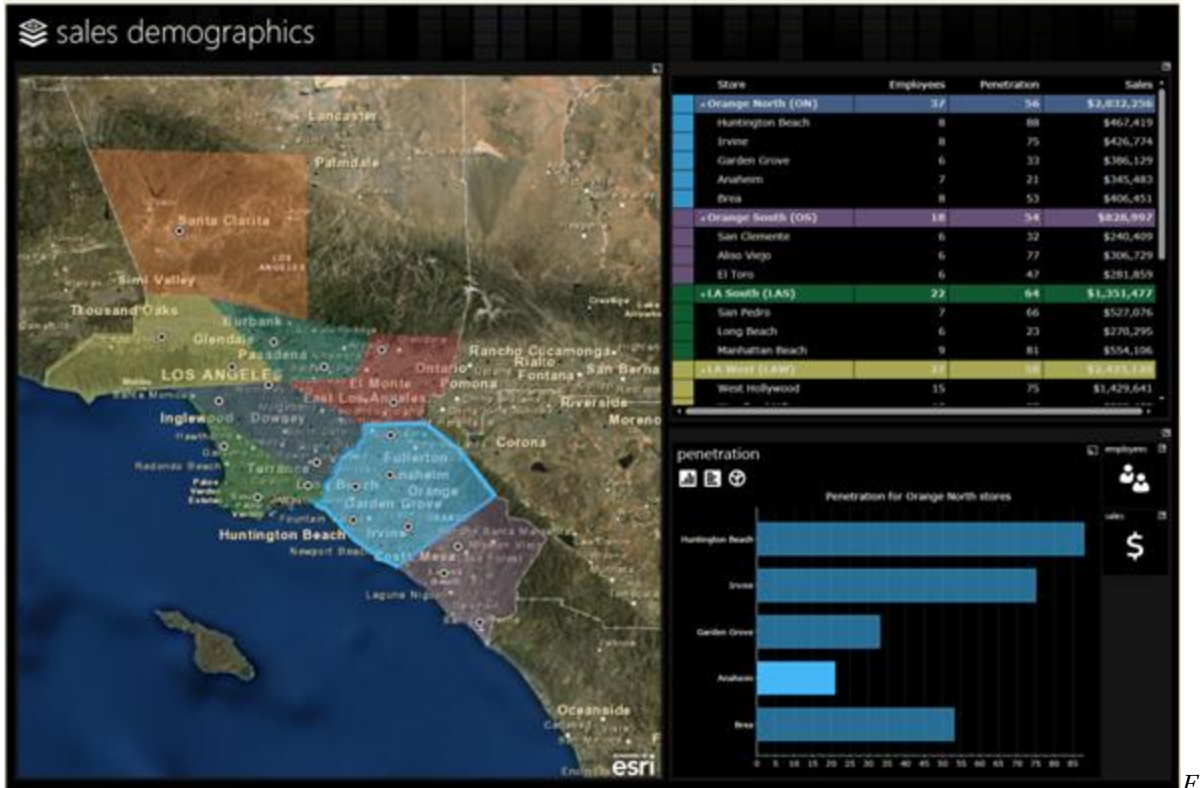
## Using Maps Powered by Esri

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.



sri 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 7  
 Windows 2008 Server

**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:

HyperPanel 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 **HyperPanel 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 HyperPanel 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 HyperPanel for WPF

To uninstall **ComponentOneStudio for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs (Programs and Features in Vista)**.
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 **CILc.exe** utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use **CILc.exe** to compile the **licenses.licx** file. The command line should look like this:  
`cilc /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 affected project.
2. Select an instance of the updated component.
3. In the Visual Studio Properties window, change any property. It does not matter which property you change; you can change it back to the previous value.
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**) and run the solution.

**Alternative 1: Follow these steps:**

1. Open a new Visual Studio.NET project.
2. Add the updated component to the form.
3. Compile and run the new project.
4. Open the licenses.licx file in the new project.
5. Copy the line that starts with the namespace of the updated component (for example, C1.Win.C1Report) and ends with a public key token.
6. Open the existing, incorrectly licensed project.
7. Open the licenses.licx file in the new project.
8. Paste the line from step 5 into this file (replace the old licensing information with the new).
9. Rebuild the project using the **Rebuild All** option (not just **Rebuild**) and run the solution.

**Alternative 2: Follow these steps:**

1. Open the affected project.
2. Delete the licenses.licx file from the project.
3. Add a new instance of the updated component to the form.
4. Rebuild and run the solution. The nag screen should not appear.
5. Remove the newly added component from the form.

Try each of these options multiple times, if necessary. If that still does not help, are you creating any of the controls in code rather than design-time? If so, you must add an entry for the control in the licenses.licx file (see <http://helpcentral.componentone.com/PrintableView.aspx?ID=1886> for more information). Also if this is a Web site, as opposed to an ASP.NET Web application, please try right-clicking the licenses.licx file and selecting **Build Runtime Licenses** from the context menu.

***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.
- **[Product Forums](#)**  
ComponentOne's [product forums](#) are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. Please note that a ComponentOne User Account is required to participate in the [ComponentOne Product Forums](#).
- **Installation Issues**  
Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the [online incident submission form](#) or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**  
Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the [Documentation team](#). Please note that e-mail sent to the [Documentation](#)

[team](#) is for documentation feedback only. [Technical Support](#) and [Sales](#) issues should be sent directly to their respective departments.

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

## Redistributable Files

**ComponentOne HyperPanel 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](mailto: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 or later. With XAML you can create a graphically

rich customized user interface, perform data binding, and much more. For more information on XAML, 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 <b>x:</b> prefix. The <b>x:</b> prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications.

When you add a `CIHyperPanel` 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:MyHP="http://schemas.componentone.com/winfx/2006/xaml"
```

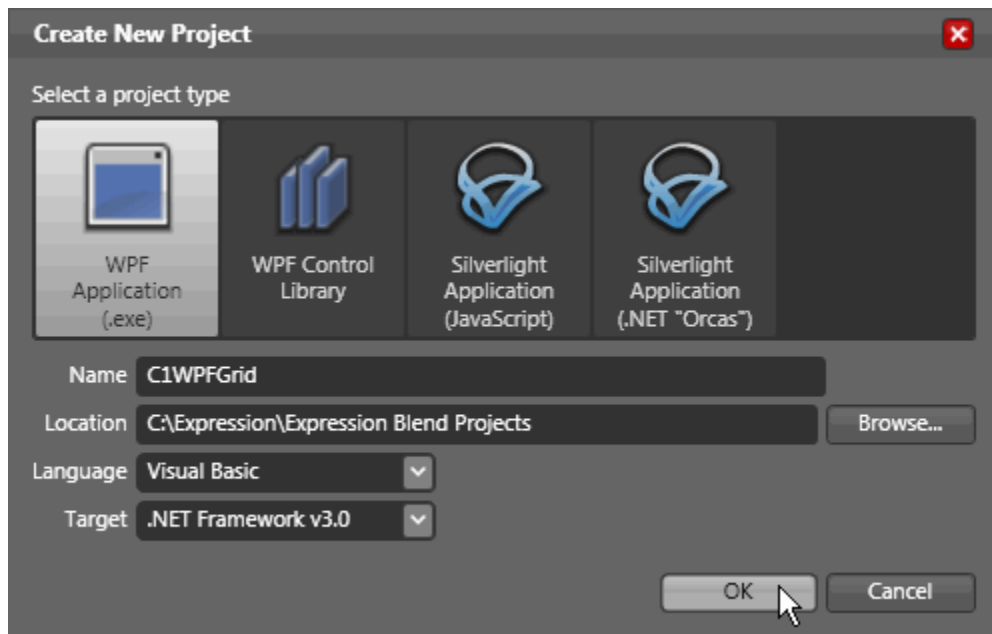
You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the panel:

```
<MyHP:CIHyperPanel Name="CIHyperPanel1" 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**.

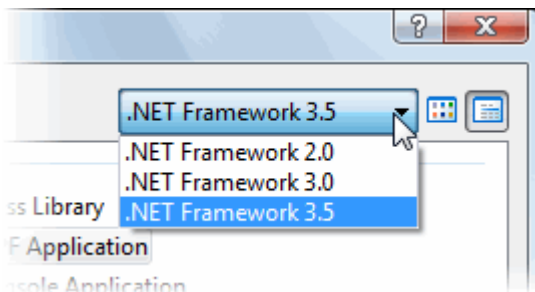


A new Blend project with a XAML window is created.

## Creating a .NET Project in Visual Studio

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

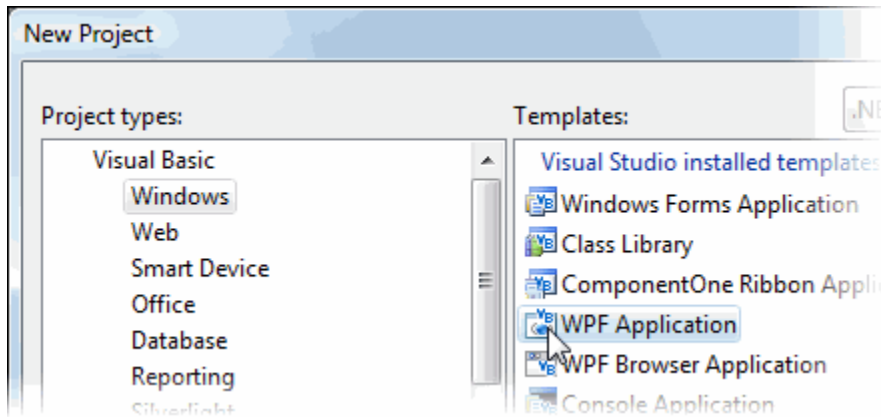
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.  
The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



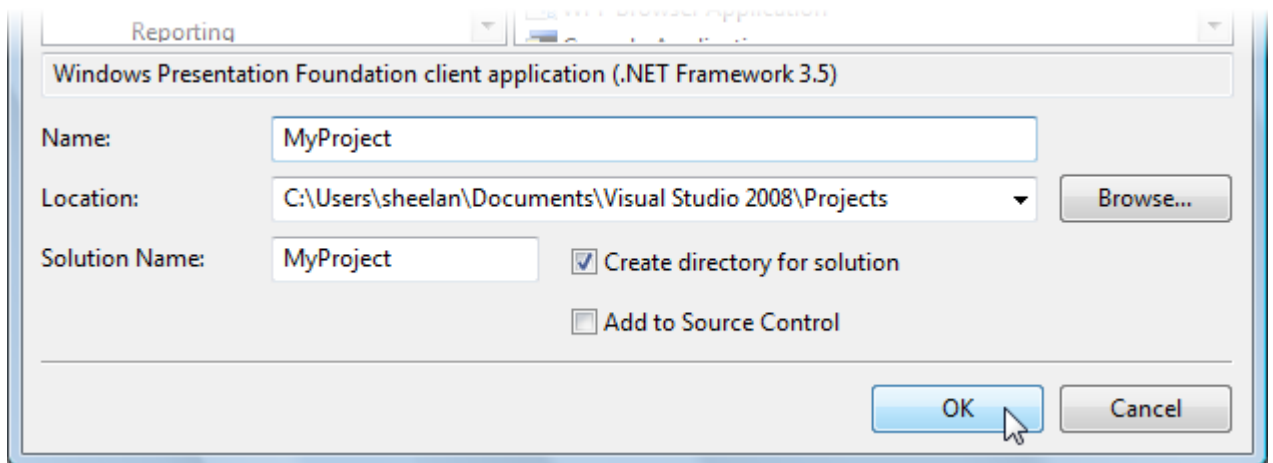
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

**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**.



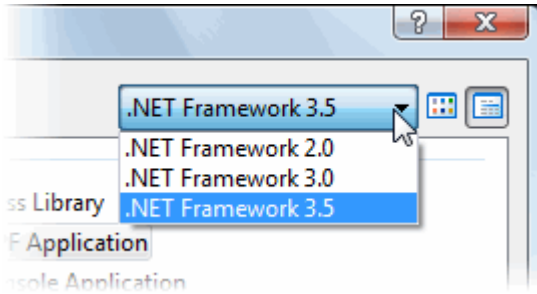
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

**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 HyperPanel for WPF Components to a Blend Project

In order to use **C1HyperPanel** or another **ComponentOne HyperPanel for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF** assembly and then add the component from Blend's **Asset Library**.


### To add a reference to the assembly:

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

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

2. 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** assembly, click the **Asset Library** button  in the Blend Toolbox. The **Asset Library** appears.
2. Click the **Controls** drop-down arrow and select **All**.
3. Select **C1HyperPanel**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1HyperPanel** component in the Toolbox to add it to **Window1.xaml**.

## Adding the HyperPanel for WPF Components to a Visual Studio Project

When you install **ComponentOne HyperPanel for WPF** the **C1HyperPanel** control should be added to your Visual Studio Toolbox. You can also manually add **ComponentOne** controls to the Toolbox.

**ComponentOne HyperPanel for WPF** provides the following control:

- **C1HyperPanel**

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

### Manually Adding HyperPanel for WPF to the Toolbox

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

- C1HyperPanel

To manually add the C1HyperPanel 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 **HyperPanel for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFHyperPanel**, 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** namespace. Note that there may be more than one component for each namespace.

### Adding HyperPanel for WPF to the Window

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

1. Add the C1HyperPanel control to the Visual Studio Toolbox.
2. Double-click C1HyperPanel or drag the control onto the window.

### Adding a Reference to the Assembly

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

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne HyperPanel 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
```

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

# Key Features

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

- **Add Dynamic Zooming**

**ComponentOne HyperPanel™ for WPF** allows you to display a large number of items in a small space. Items far from the mouse are shrunk and don't take up much space.

- **Control the Zoom Effect**

Determine how much zooming should be applied when the mouse moves over the panel. Use the **Distribution** property to control the strength of the zoom effect.

- **Limit the Zoom Effect**

Use the **MinElementScale** property to prevent items from getting too small.

- **Control Item Opacity**

Use the **ApplyOpacity** property to make elements near the mouse opaque. Items far from the mouse become more transparent, conveying an idea of distance.

# HyperPanel for WPF Quick Start

The following quick start guide is intended to get you up and running with **HyperPanel for WPF**. In this quick start you'll start in Visual Studio and create a new project, add a **HyperPanel for WPF** panel to your application, add controls within C1HyperPanel, and customize the appearance and behavior of the panel.

You will create a simple application using a C1HyperPanel panel that contains several items. You'll then customize the C1HyperPanel panel's appearance and behavior settings to explore the possibilities of using **HyperPanel for WPF**.

## Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a WPF application using **HyperPanel for WPF**. When you add a C1HyperPanel panel to your application and items to the panel you'll have a completely interactive way of viewing and selecting items.

To set up your project, complete the following steps:

1. Create a new WPF project in Visual Studio. For more information about creating a WPF project, see [Creating a .NET Project in Visual Studio](#) (page 12).
2. Add the follow XAML markup just under the <Window> tag:

- XAML to Add

```
<Window.Resources>
  <ResourceDictionary>
    <Style x:Key="letterStyle" TargetType="Border">
      <Setter Property="Height" Value="60" />
      <Setter Property="Width" Value="60" />
      <Setter Property="Margin" Value="2" />
      <Setter Property="CornerRadius" Value="3" />
      <Setter Property="BorderThickness" Value="4" />
      <Setter Property="BorderBrush" >
        <Setter.Value>
          <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
            <GradientStop Color="#FFBDBDBD"/>
            <GradientStop Color="#FFDADADA" Offset="0.5"/>
            <GradientStop Color="#FFBDBDBD" Offset="1"/>
          </LinearGradientBrush>
        </Setter.Value>
      </Setter>
    </Style>
    <DataTemplate x:Key="letterTemplate" >
      <Border Background="Blue" Style="{StaticResource letterStyle}"
>
        <TextBlock Text="{Binding}" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="32" TextAlignment="Center"
TextWrapping="Wrap" >
          <TextBlock.Foreground>
            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
              <GradientStop Color="BlueViolet"/>
              <GradientStop Color="Thistle"
Offset="0.4"/>
            </LinearGradientBrush>
          </TextBlock.Foreground>
        </TextBlock>
      </Border>
    </DataTemplate>
  </ResourceDictionary>
</Window.Resources>
```

```

        <GradientStop Color="AntiqueWhite"
Offset="0.6"/>
        <GradientStop Color="White" Offset="1"/>
    </LinearGradientBrush>
    </TextBlock.Foreground>
</TextBlock>
</Border>
</DataTemplate>
</ResourceDictionary>
</Window.Resources>

```

This will add a style to format the content we will add to the C1HyperPanel panel.

3. Navigate to the Toolbox and double-click the **C1HyperPanel** icon to add the panel to Window1.
4. Resize window and the C1HyperPanel panel to fill the window. Later you'll add items to the C1HyperPanel panel.

You've successfully created a WPF application and added a C1HyperPanel panel to the application. In the next step you'll add controls to and customize C1HyperPanel.

## Step 2 of 3: Adding Content to the Panel

In the previous step you created a new WPF project and added a C1HyperPanel panel to the application. In this step you'll continue by adding controls to the panel.

Complete the following steps:

1. Switch to XAML view and navigate to just under the `<Window>` tag.
2. In the Design pane, click once inside the C1HyperPanel panel; now when you add items they will be added inside the panel.
3. Navigate to the Toolbox and double-click the **ContentControl** icon to add the control to C1HyperPanel. That's all you have to do to add an item to a C1HyperPanel panel – you can items as you would normally to other panels include the **Grid** and **Canvas**.
4. Switch to XAML view and update the **ContentControl**'s markup so that it appears similar to the following:

```

<ContentControl Content="h" ContentTemplate="{StaticResource
letterTemplate}"/>

```

5. Enter the following XAML under the **ContentControl**'s markup and within the C1HyperPanel to add additional controls to the panel:

- XAML to Add

```

<ContentControl Content="e" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="o" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content=" " ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="w" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="o" ContentTemplate="{StaticResource
letterTemplate}"/>

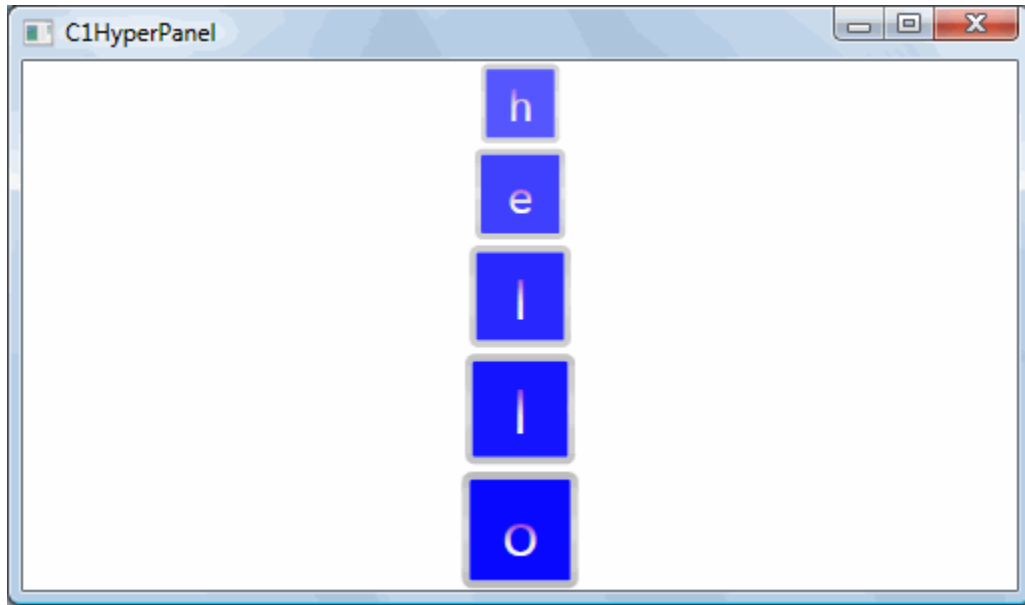
```

```

<ContentControl Content="r" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="d" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="!" ContentTemplate="{StaticResource
letterTemplate}"/>

```

- From the **Project** menu, select **Test Solution** to view how your application will appear at run time. It will appear similar to the following:



Notice that you can move your mouse over the items in the panel and they will appear to move.

You've successfully created a WPF application and added C1HyperPanel and controls to the application. In the next step you'll customize C1HyperPanel.

## Step 3 of 3: Customizing the Application

In the previous steps you created a new WPF project and added a C1HyperPanel panel and several **ContentControls** to the application. In this step you'll continue by setting properties to customize those controls.

Complete the following steps:

- In Design view, click once on the **C1HyperPanel** control to select it.
- Navigate to the Properties window and set the Orientation property to **Horizontal**.

The Orientation property determines if items in the panel are displayed horizontally or vertically. By default Orientation is set to **Vertical** and the panel displays content vertically; setting Orientation property to **Horizontal** will display content horizontally.

- In the Properties window, set the Distribution property to "0.2".

The Distribution property consists of a number between .1 and 1.0 and controls how elements are zoomed near the center of the panel. The smaller the value, the more visible the zoom effect. By default, the

property is set to "0.5". Setting Distribution to ".02" will cause elements in the center to appear even more zoomed in than elements at the edge of the panel.

4. Set the MinElementScale property to "0.5".

The MinElementScale property consists of a number between 0 and 1.0 and determines how small elements near the edge of the panel will appear when compared to elements near the center. By default, the property is set to "0". Setting MinElementScale will ensure that elements near the edge of the panel will appear half their original size at the smallest.

5. Set the Center property to "0.1".

The Center property consists of a number between 0 and 1.0 and sets where the center, or the most zoomed in element, of the panel is when the application is initially run. By default, the property is set to "0.5" and the center is in the middle. Setting Center will move the initially zoomed element to the left side of the control. This value is updated automatically as the mouse moves over the panel at run time.

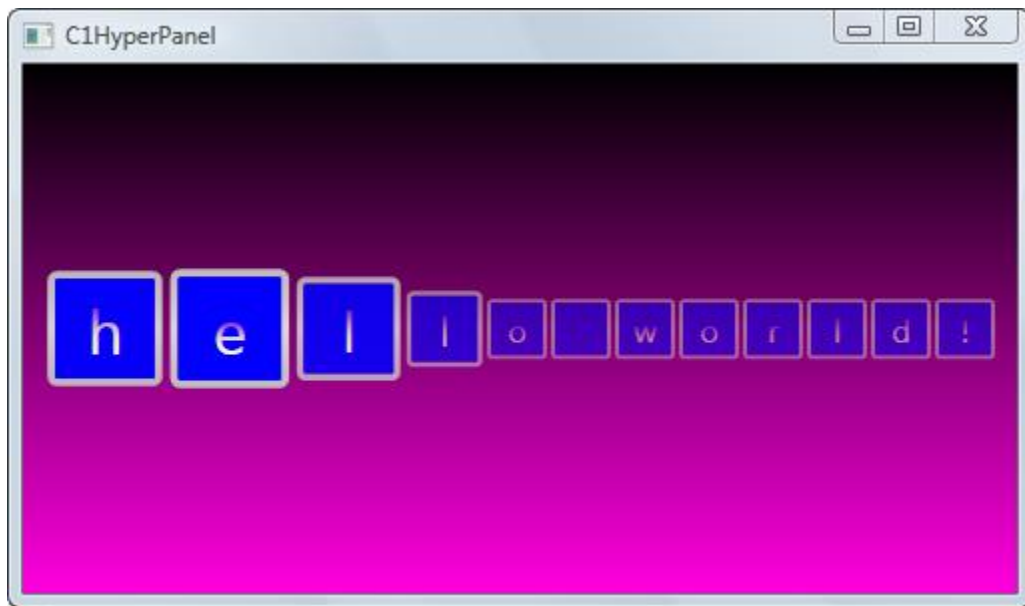
6. Switch to XAML view and add the following markup just after the <C1HyperPanel> tag:

```
<c1:C1HyperPanel.Background>
  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
    <GradientStop Color="#FF000000" Offset="0"/>
    <GradientStop Color="#FFFF00DF" Offset="1"/>
  </LinearGradientBrush>
</c1:C1HyperPanel.Background>
```

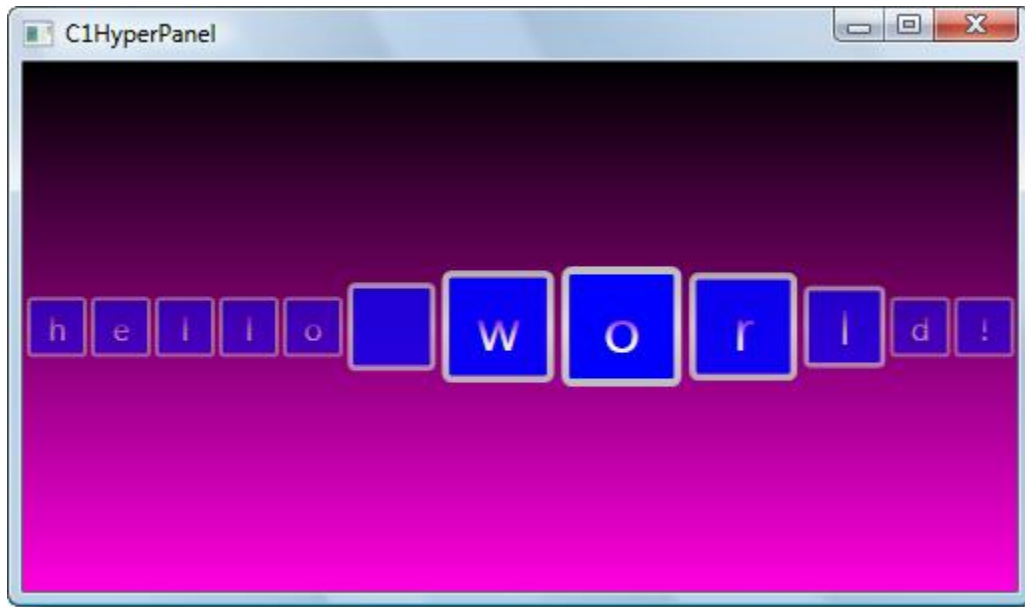
This markup will add a gradient background to the C1HyperPanel.

7. From the **Project** menu, select **Test Solution** to view how your application will appear at run time.

The application will initially appear similar to the following:



8. Move your mouse in the panel and notice that the Center of the content changes:



Congratulations! You've completed the **HyperPanel for WPF** quick start and created a **ComponentOne HyperPanel for WPF** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.



# Working with HyperPanel for WPF

**ComponentOne HyperPanel for WPF** includes the **C1HyperPanel** panel, a simple **StackPanel** which provides an automatic zoom effect for items near the mouse. When you add the **C1HyperPanel** panel to a XAML window, it exists as an empty panel where you can add controls and content as you would to any other panel, the **Grid** or **Canvas**.

## Basic Properties

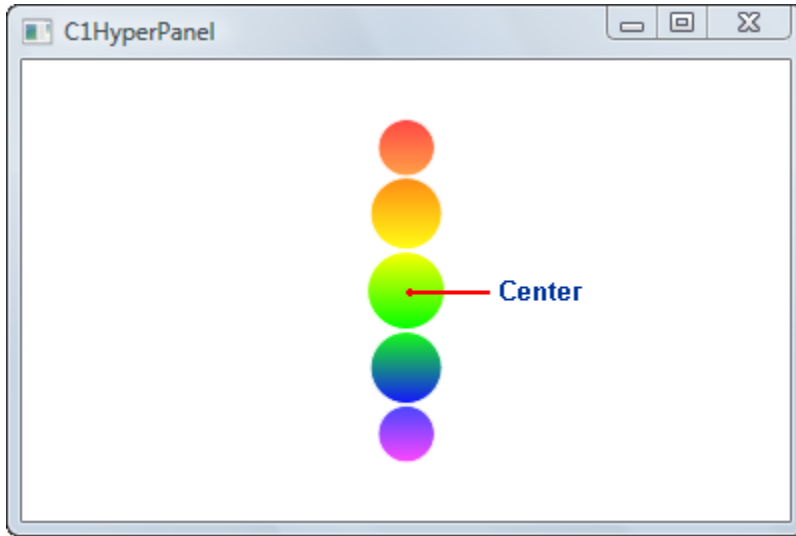
**ComponentOne HyperPanel for WPF** includes several properties that allow you to set the functionality of the panel. Some of the more important properties that vary from the standard **StackPanel** properties are listed below.

The following properties let you customize the **C1HyperPanel** panel:

Property	Description
ApplyOpacity	Gets or sets a Boolean value that determines if opacity is applied to elements away from the center of the panel.
Center	Gets or sets the center of the <b>C1HyperPanel</b> as a percentage of the control size.
Distribution	Gets or sets a value between 0.1 and 1.0 that controls how much zooming should be applied to elements near the center.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the panel's content.
MinElementScale	Gets or sets a value between zero and one that determines the minimum scale to be applied to elements when they are away from the center.
Orientation	Gets or sets a value that indicates the dimension by which child elements are stacked.
VerticalContentAlignment	Gets or sets the vertical alignment of the panel's content.

## Zoom Center

The **Center** property determines where the initial center of the panel is located and indicates the largest element of the zoom effect of the panel. By default, the center is at the exact center of content in the panel:

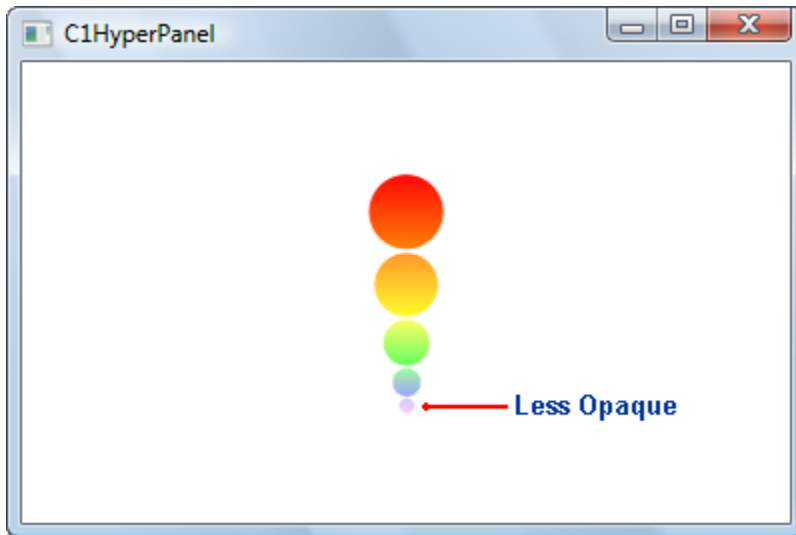


The Center can be set to a value from 0 and 1. By default, the Center is "0.5". If you set Center to "0", the Center will be the top edge of the panel (or left edge if the Orientation is set to **Horizontal**).

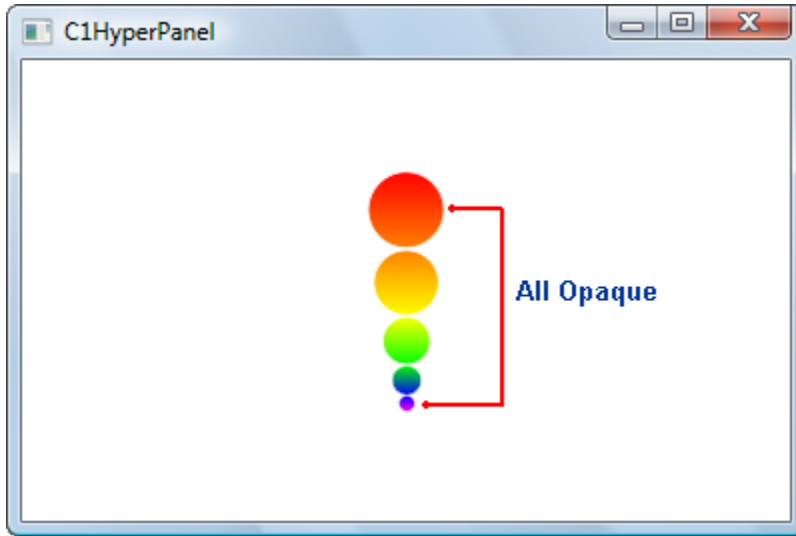
When Orientation is set to **Horizontal**, the Center property is also determined by the [FlowDirection](#) property. If Center is set to "0" and [FlowDirection](#) is set to **LeftToRight** (default) then the Center will be the left edge of the panel. If the [FlowDirection](#) is set to **RightToLeft** the Center will be the right edge of the panel.

## Edge Opacity

The ApplyOpacity property lets you control how items further from the Center of the panel appear. If ApplyOpacity is **True** (default) then items further away from the center will be less opaque:

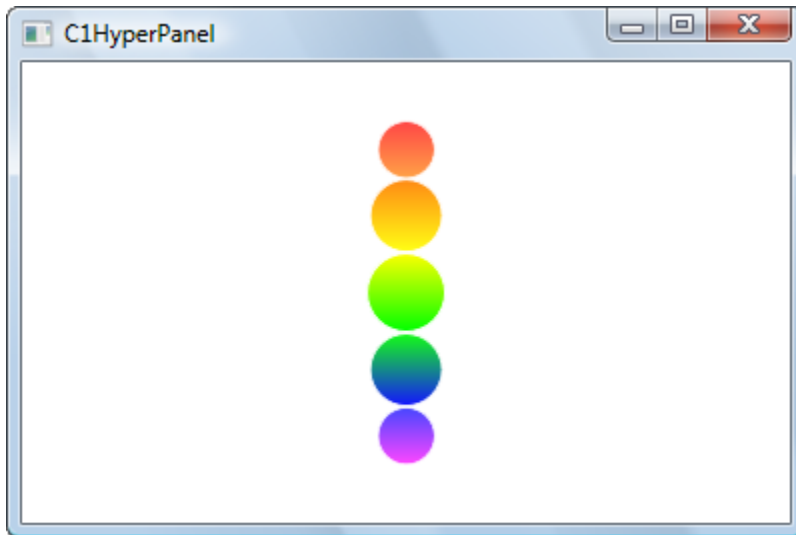


If ApplyOpacity is **False** then items further away from the center will appear at the same opacity level as the item in the center:

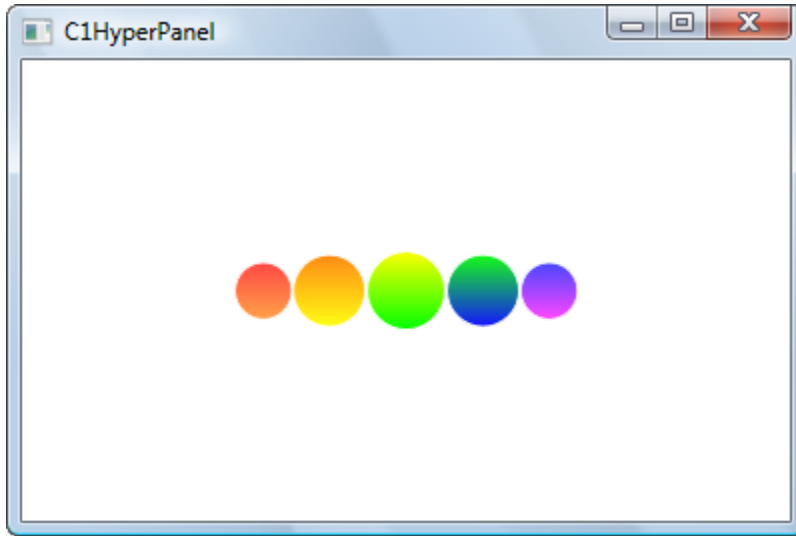


## Horizontal and Vertical Orientation

The Orientation property determines how content is laid out within the C1HyperPanel panel. By default, Orientation is set to **Vertical** and content appears stacked from top to bottom vertically in the panel:

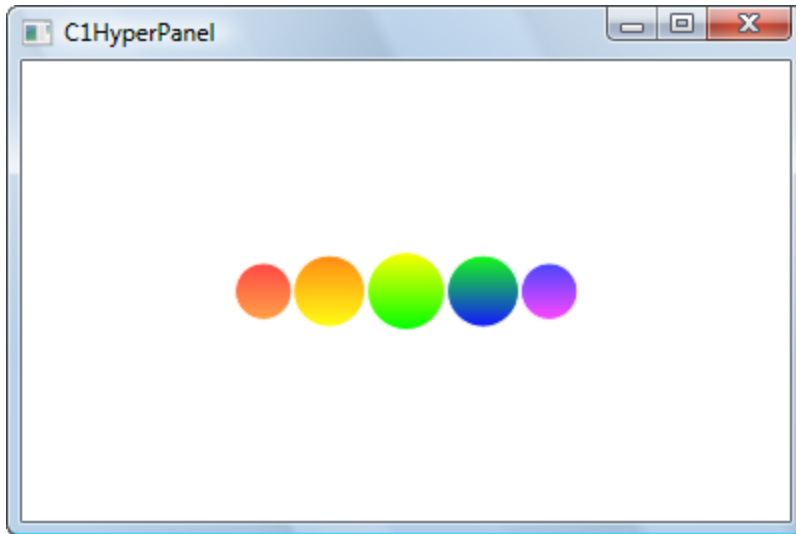


When Orientation is set to **Horizontal** content will appear stacked from left to right horizontally in the panel:

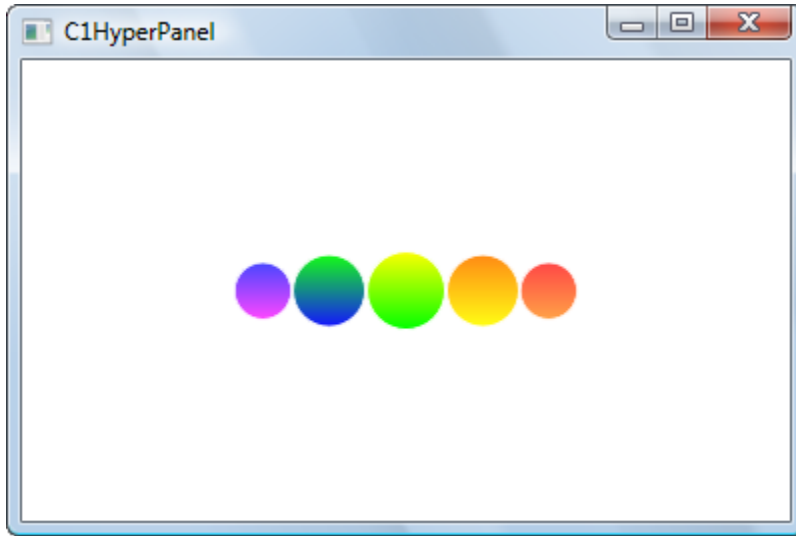


## Flow Direction

The [FlowDirection](#) property determined how content flows when the Orientation property is set to **Horizontal**. By default, [FlowDirection](#) is set to **LeftToRight** and when the Orientation property is set to **Horizontal**, content will appear stacked from left to right across the panel:



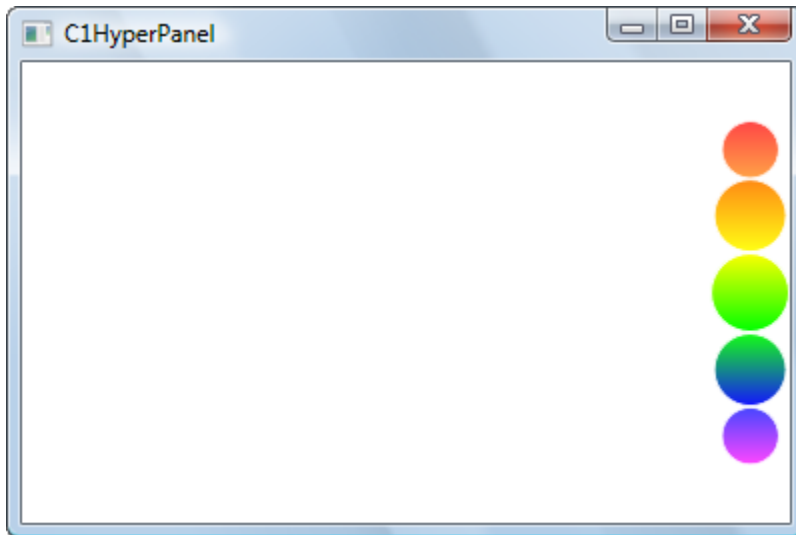
If [FlowDirection](#) is set to **LeftToRight** and the Orientation property is set to **Horizontal**, content will instead appear stacked from right to left across the panel:



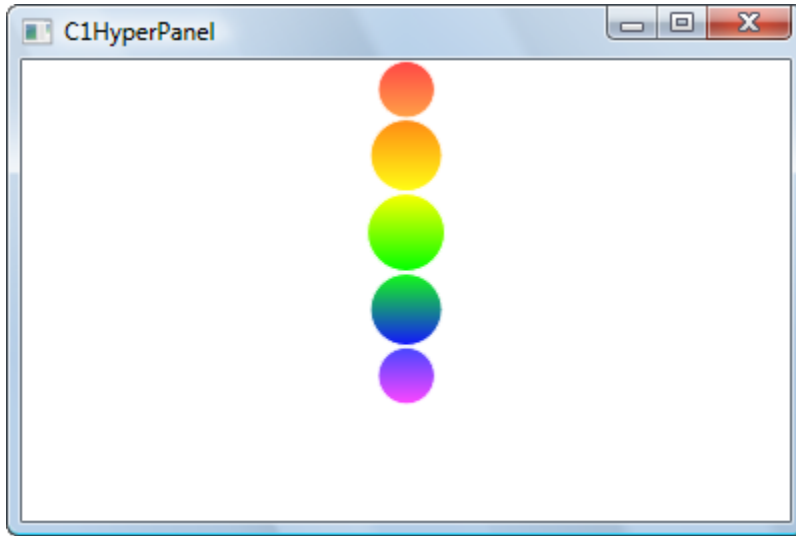
## Alignment

The **HorizontalAlignment** and **VerticalAlignment** properties control how the C1HyperPanel panel is aligned in the containing panel or window. By default, both properties are set to **Stretch** and the panel is stretched to fill the entire space available.

**HorizontalAlignment** options include **Left**, **Center**, **Right**, and **Stretch**. If, for example, **HorizontalAlignment** was set to **Right**, the panel would appear at the right of the available area like in the following image:



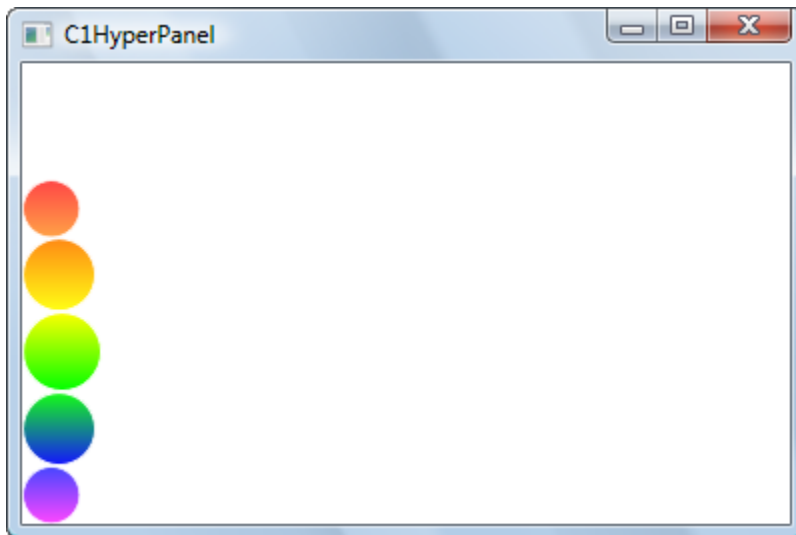
**VerticalAlignment** options include **Top**, **Center**, **Bottom**, and **Stretch**. If, for example, **VerticalAlignment** was set to **Top**, the panel would appear at the top of the available area like in the following image:



## Content Alignment

The `HorizontalAlignment` and `VerticalContentAlignment` properties control how content in the `C1HyperPanel` panel is aligned within the panel. By default, both properties are set to **Center** and content is centered within the panel.

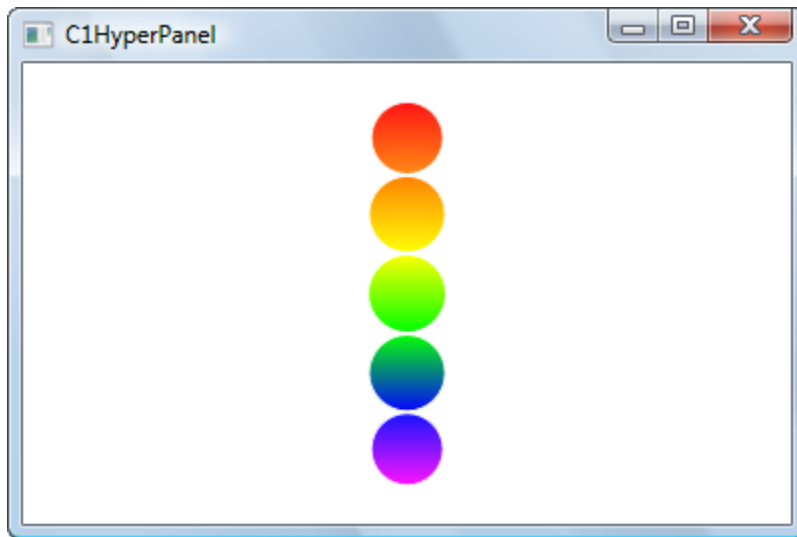
`HorizontalAlignment` options include **Left**, **Center**, **Right**, and **Stretch**. `VerticalContentAlignment` options include **Top**, **Center**, **Bottom**, and **Stretch**. If, for example, `HorizontalAlignment` was set to **Left** and `VerticalContentAlignment` was set to **Bottom**, the panel would appear in the bottom-left corner of the panel like in the following image:



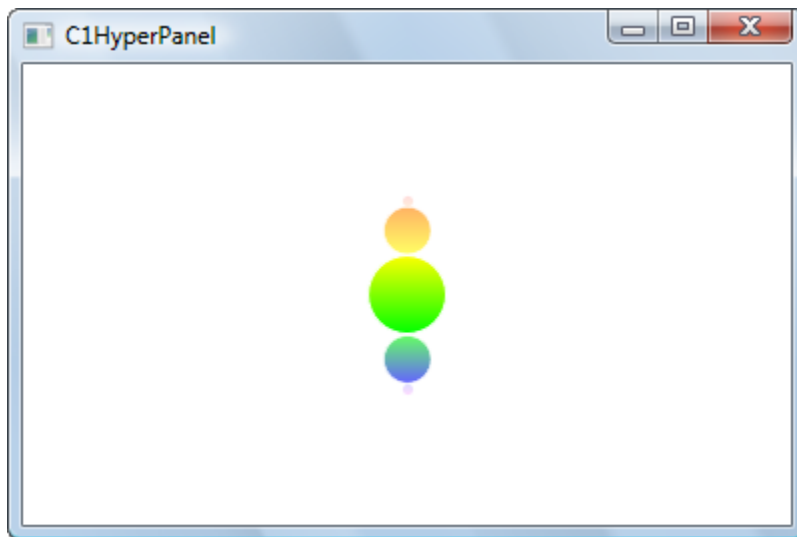
## Distribution

The `Distribution` property gets or sets a value between 0.1 and 1.0 that controls how much zooming should be applied to elements near the center of the panel. The smaller the value, the more distant items away from the center of the panel will appear. By default `Distribution` is set to "0.5".

If Distribution is set to "1", all elements will appear at the same zoom level, for example in the following image:



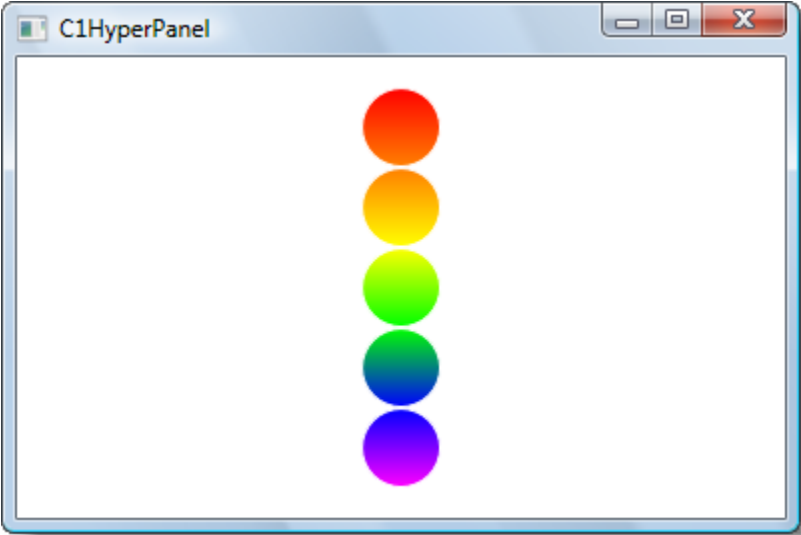
If Distribution is set to a smaller number, such as "0.2", elements away from the center will appear further zoomed out:



## Scale

The `MinElementScale` property gets or sets a value that determines the minimum scale to be applied to elements when they are away from the center. If `MinElementScale` is set to "0.1", items furthest away from the center will appear at smallest 10% of their original size.

By default, `MinElementScale` is set to "0" and items furthest away from the center of the panel appear completely zoomed out. The larger the number, the more zoomed in elements further from the center appear. For example, when `MinElementScale` is set to "1" as in the image below, all items appear to be the same distance away and elements appear at 100% of their original size. When the mouse is moved over elements in the panel they are not zoomed in or out at all and appear static:



# HyperPanel for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with the ComponentOne Studios. Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

## C# Samples

The following C# sample is included:

Sample	Description
ControlExplorer	The <b>HyperPanel</b> page in the <b>ControlExplorer</b> sample demonstrates how to add content to and customize the C1HyperPanel container.

# HyperPanel 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 C1HyperPanel panel in general. If you are unfamiliar with the **ComponentOne HyperPanel for WPF** product, please see the [HyperPanel for WPF Quick Start](#) (page 17) first.

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

Each task-based help topic also assumes that you have created a new WPF project and added a C1HyperPanel panel to your project. For additional information on this topic, see [Creating a .NET Project in Visual Studio](#) (page 12) or [Creating a Microsoft Blend Project](#) (page 11).

## Adding a Control to the Panel

By default the C1HyperPanel panel appears empty and contains no content. Adding controls and other content to the panel is as easy as adding content to any other panel, **Canvas**, or **Grid**. In the following steps you'll add a button to your C1HyperPanel.

### At Design Time

To add a button to C1HyperPanel at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Visual Studio Toolbox and double-click the **Button** control. The control will be added to the panel.

### In XAML

To add a button to C1HyperPanel add the `<Button>` tag after the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1">  
    <Button Height="50" Name="button1" Width="50"></Button>  
</c1:C1HyperPanel>
```

### In Code

To add a button to C1HyperPanel, double-click the window to switch to Code view and add the **Window\_Loaded** event handler; then add code so it appears like the following:

- Visual Basic

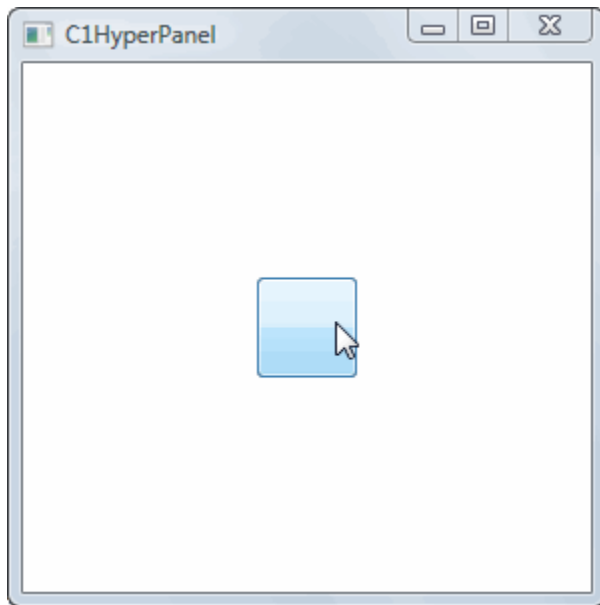
```
Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim button1 = New Button
    button1.Height = 50
    button1.Width = 50
    Me.C1HyperPanel1.Children.Add(button1)
End Sub
```

- C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    Button button1 = new Button();
    button1.Height = 50;
    button1.Width = 50;
    this.c1HyperPanel1.Children.Add(button1);
}
```

**Run your project and observe:**

The C1HyperPanel panel will appear with a button:



## Changing the Background Color

By default the C1HyperPanel panel appears transparent and all elements below the panel will show through. If you choose, you can set the color of the panel using the **Background** property. For example, the following steps will detail how to change the color of the background to black.

### At Design Time

To change **C1HyperPanel**'s background color to black at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.

2. Navigate to the Properties window and locate the **Background** property.
3. Select the drop-down arrow next to the property name and select **Black** from the list.

### In XAML

To change **C1HyperPanel**'s background color to black in XAML add `Background="Black"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Background="Black">
```

### In Code

For example, to change the background color add the following code to your project:

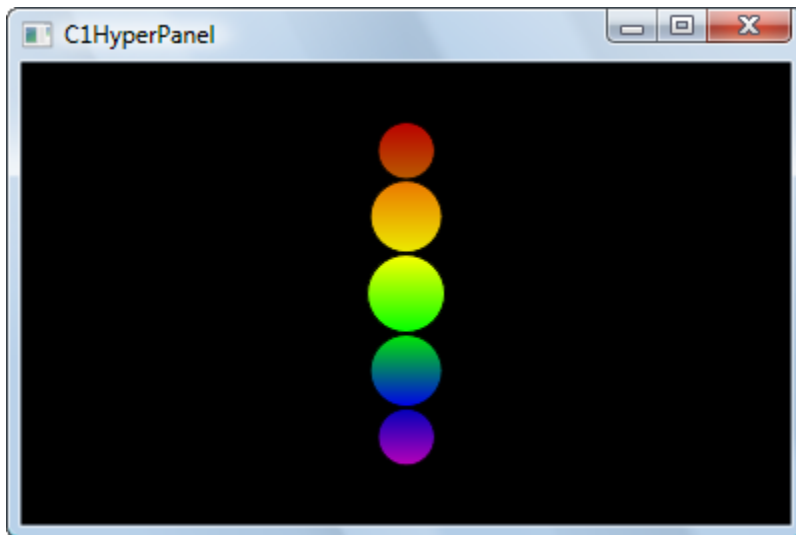
- Visual Basic  

```
Me.C1HyperPanel1.Background = System.Windows.Media.Brushes.Black
```
- C#  

```
this.c1HyperPanel1.Background = System.Windows.Media.Brushes.Black;
```

### Run your project and observe:

The C1HyperPanel panel will appear with a black background:



## Setting the Start Position

The Center property determines where the initial center of the panel is located. The "center" here does not actually mean the actual center of the panel, but rather indicates the largest element of the zoom effect of the panel. For example, the following steps will detail how to change the Center of the panel to the first item.

### At Design Time

To change **C1HyperPanel**'s Center at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties window and locate the Center property.
3. Click in the text box next to the Center property and enter "0".

### In XAML

To change **C1HyperPanel**'s Center in XAML add `Center="0"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Center="0">
```

### In Code

To change the Center in code, add the following code to your project:

- Visual Basic

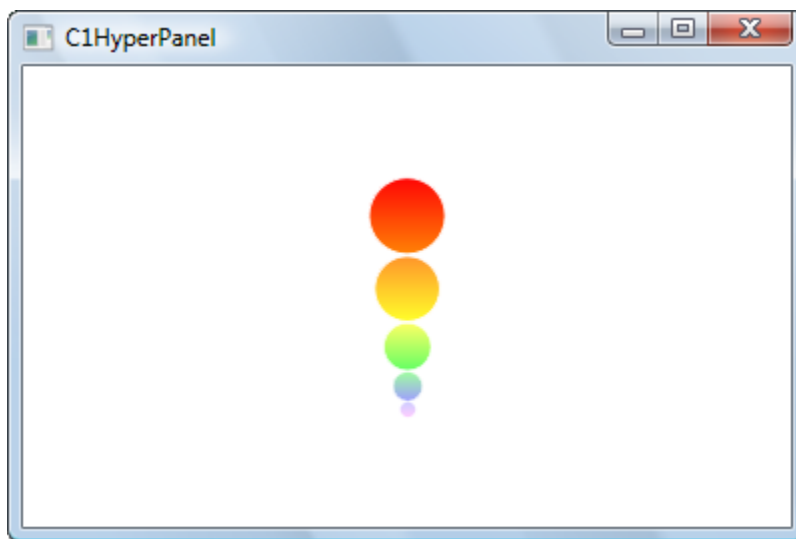
```
Me.C1HyperPanel1.Center = "0"
```

- C#

```
this.c1HyperPanel1.Center = "0";
```

### Run your project and observe:

The C1HyperPanel panel will appear centered on the first item in the panel:



## Changing the Scale

The `MinElementScale` property determines how small elements farthest away from the Center of the `C1HyperPanel` will appear. For example, using `MinElementScale` you can choose to set all elements to the same size which would remove the zoom effect.

### At Design Time

To set all `C1HyperPanel` elements to the same size at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties window and locate the `MinElementScale` property.
3. Click in the text box next to the `MinElementScale` property and enter "1".

### In XAML

To set all `C1HyperPanel` elements to the same size in XAML add `MinElementScale="1"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" MinElementScale="1">
```

### In Code

To set all C1HyperPanel elements to the same size in code, add the following code to your project:

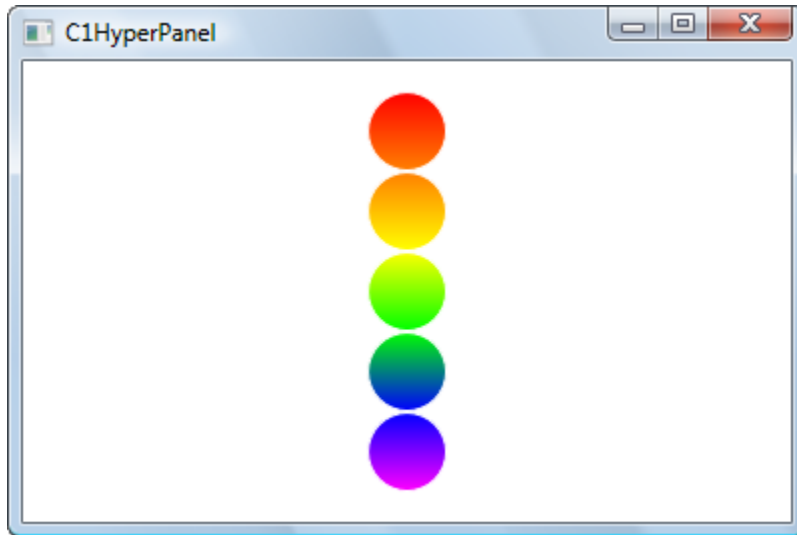
- Visual Basic  

```
Me.C1HyperPanel1.MinElementScale = "1"
```
- C#  

```
this.c1HyperPanel1.MinElementScale = "1";
```

### Run your project and observe:

All items in the C1HyperPanel panel will appear the same size and when you mouse over items there will be no zoom effect:



## Setting the Orientation

By default the Orientation property is set to **Vertical** and content appears stacked from top to bottom vertically in the panel. If you choose, you can change the Orientation so that content appears stacked horizontally instead.

### At Design Time

To set the Orientation so that content appears stacked horizontally at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties window and locate the Orientation property.
3. Select the drop-down arrow next to the Orientation property and choose **Horizontal**.

### In XAML

To set the Orientation so that content appears stacked horizontally in XAML add `Orientation="Horizontal"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Orientation="Horizontal">
```

### In Code

To set the Orientation so that content appears stacked horizontally in code, add the following code to your project:

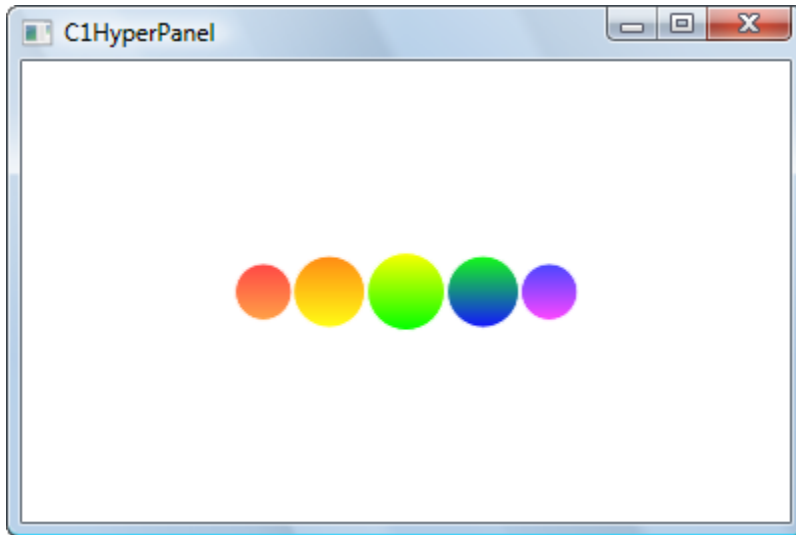
- Visual Basic  

```
Me.C1HyperPanel1.Orientation = Orientation.Horizontal
```
- C#

```
this.clHyperPanel1.Orientation = Orientation.Horizontal;
```

### Run your project and observe:

When Orientation is set to **Horizontal** content will appear stacked from left to right horizontally in the panel:



## Setting Element Distribution

The Distribution property controls how much zooming should be applied to elements near the center of the panel. The smaller the value, the smaller items away from the center of the panel will appear. In the following steps you'll set the Distribution so that items away from the center appear more zoomed out.

### At Design Time

To set the Distribution so that items away from the center appear more zoomed out at design time, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties window and locate the Distribution property.
3. Click in the text box next to the Distribution property and enter "0.2".

### In XAML

To set the Distribution so that items away from the center appear more zoomed out in XAML add `Distribution="0.2"` to the `<cl:C1HyperPanel>` tag so that it appears similar to the following:

```
<cl:C1HyperPanel Name="C1HyperPanel1" Distribution="0.2">
```

### In Code

To set the Distribution so that items away from the center appear more zoomed out in code, add the following code to your project:

- Visual Basic  

```
Me.C1HyperPanel1.Distribution = "0.2"
```
- C#  

```
this.clHyperPanel1.Distribution = "0.2";
```

### Run your project and observe:

With the Distribution property set to a smaller number, elements away from the center appear further zoomed out:

