

---

ComponentOne

# ColorPicker 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 ColorPicker for WPF Overview .....	1
Installing ColorPicker for WPF .....	1
Studio for WPF Setup Files.....	1
Using Maps Powered by Esri .....	2
System Requirements .....	3
Installing Demonstration Versions.....	4
Uninstalling ColorPicker for WPF.....	4
End-User License Agreement .....	4
Licensing FAQs .....	4
What is Licensing?.....	5
How does Licensing Work?.....	5
Common Scenarios .....	6
Troubleshooting.....	8
Technical Support .....	9
Redistributable Files.....	10
About this Documentation.....	10
XAML and XAML Namespaces.....	11
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 ColorPicker for WPF Components to a Blend Project .....	14
Adding the ColorPicker for WPF Components to a Visual Studio Project .....	14
Key Features .....	15
ColorPicker for WPF Quick Start .....	19
Step 1 of 4: Setting up the Application.....	19
Step 2 of 4: Adding C1ColorPicker Controls .....	20
Step 3 of 4: Adding Code to the Application .....	21
Step 4 of 4: Running the Application.....	22
Working with ColorPicker for WPF .....	25
Basic Properties.....	25

Basic Events .....	26
ColorPicker Mode.....	26
Basic ColorPicker Mode.....	26
Advanced ColorPicker Mode.....	27
Additional Controls .....	28
C1SpectrumColorPicker.....	28
C1HexColorBox.....	29
C1CheckedBorder .....	30
Available ColorPicker Palettes.....	30
Recent Colors.....	32
Drop-Down Direction.....	32
ColorPicker Layout and Appearance .....	32
Layout in a Panel.....	33
ColorPicker Appearance Properties.....	33
Color Properties.....	33
Alignment Properties.....	33
Border Properties.....	34
Size Properties .....	34
ComponentOne ClearStyle Technology .....	34
How ClearStyle Works.....	35
ClearStyle Properties .....	35
ColorPicker Templates.....	35
ColorPicker Styles .....	36
ColorPicker Template Parts .....	36
ColorPicker Visual States.....	38
XAML Elements.....	39
ColorPicker for WPF Samples.....	41
ColorPicker for WPF Task-Based Help .....	41
Setting the Palette.....	41
Creating a Custom Palette .....	42
Changing the Background Color.....	44
Changing the Drop-Down Window Direction.....	45
Hiding Recent Colors.....	46

# ComponentOne ColorPicker for WPF Overview

**ComponentOne ColorPicker™ for WPF** is a color input editor that provides a rich, interactive color selection interface. Users can select colors from professionally designed palettes or custom colors that you have chosen.

You can choose to include a basic color palette with preselected and standard colors, an advanced palette that users can use to completely customize their color selection, or both! **ColorPicker for WPF** even includes transparency, hexadecimal color, and RGB and HLS color model support to provide a rich visual color input interface.

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 19)
- [Task-Based Help](#) (page 41)

## Installing ColorPicker for WPF

The following sections provide helpful information on installing **ComponentOne ColorPicker 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 ColorPicker 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
- C1.WPF.Extended.dll
- C1.WPF.Extended.Expression.Design.dll
- C1.WPF.Extended.VisualStudio.Design.dll
- C1.WPF.Extended.Expression.Design.4.dll
- C1.WPF.Extended.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 C1ColorPicker styles and templates which can be used for creating your own custom styles and templates.

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

**H2Help** Contains Microsoft Help 2.0 integrated documentation for all Studio components.

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

## Samples

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

**Windows XP path:** C:\Documents and Settings\\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 **Studio for WPF**.

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.

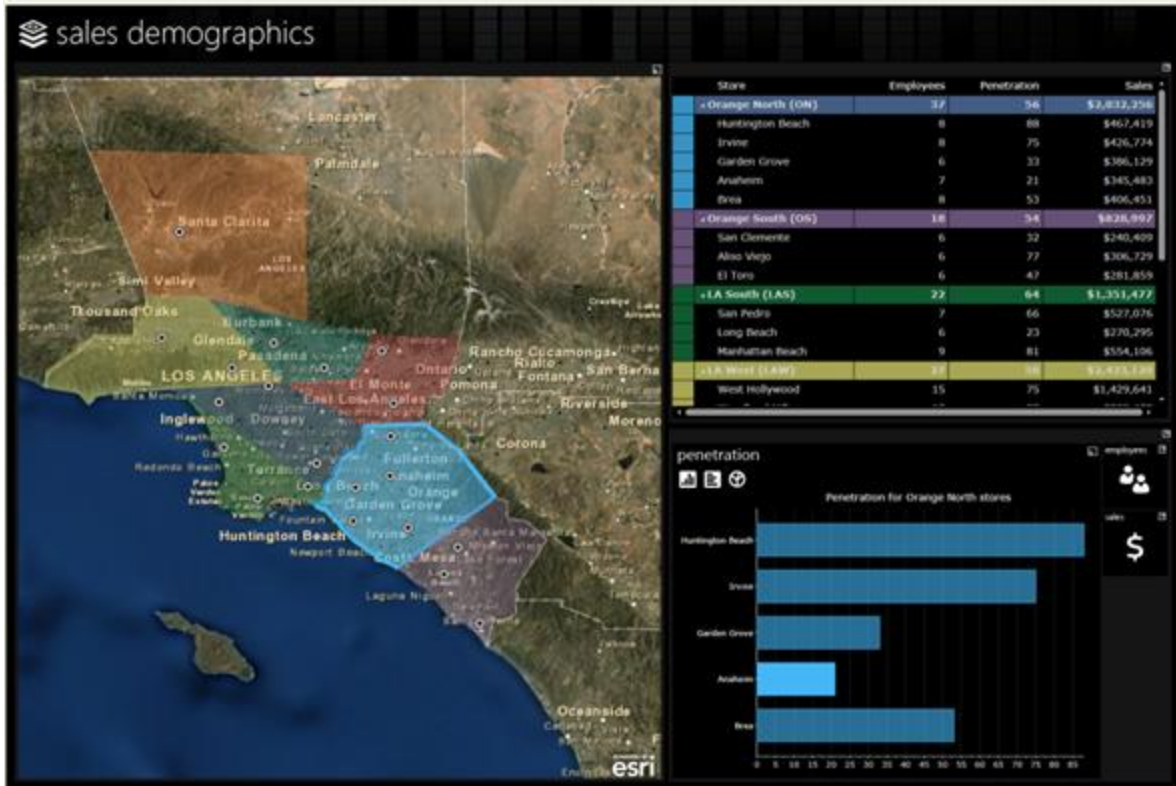
## Using Maps Powered by Esri

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

**Microsoft® Expression®  
Blend Compatibility:** **ColorPicker 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 **ColorPicker 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 ColorPicker 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 ColorPicker for WPF

To uninstall **ComponentOne ColorPicker for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs (Programs and Features)** in Windows 7/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 ColorPicker 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](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 `C1ColorPicker` 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` and the namespace is `C1.WPF`. 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:MyNB="clr-namespace:C1.WPF;assembly=C1.WPF"
```

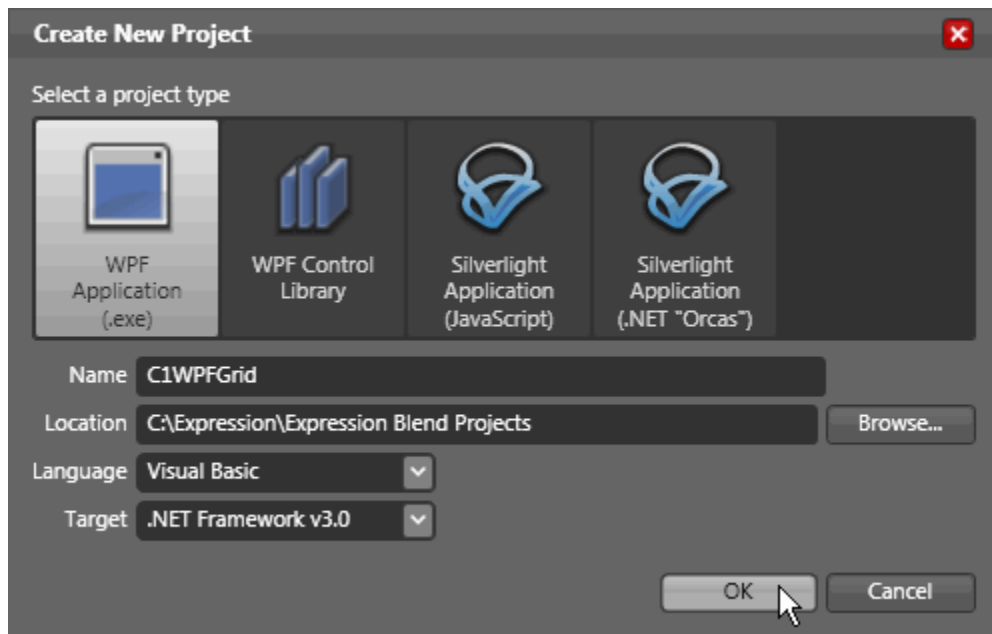
You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the `ColorPicker`:

```
<MyNB:C1ColorPicker Name="C1ColorPicker1" 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**.

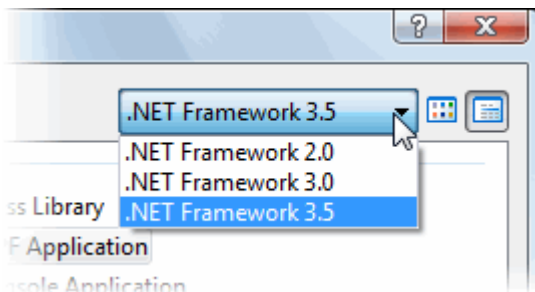


A new Blend project with a XAML window is created.

## Creating a .NET Project in Visual Studio

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

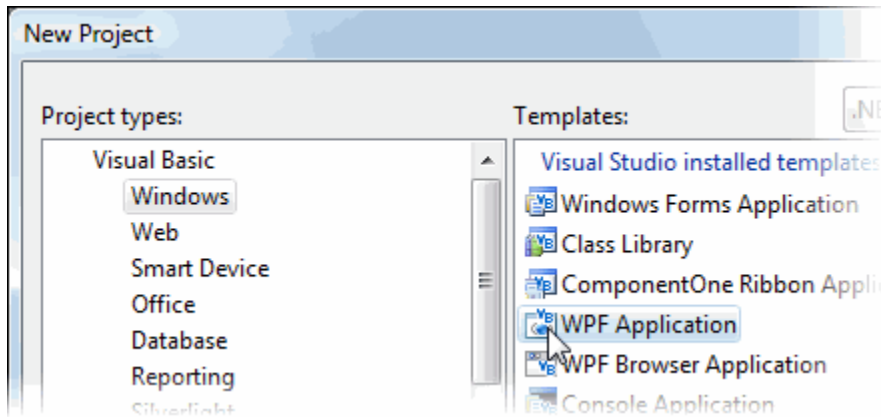
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.  
The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



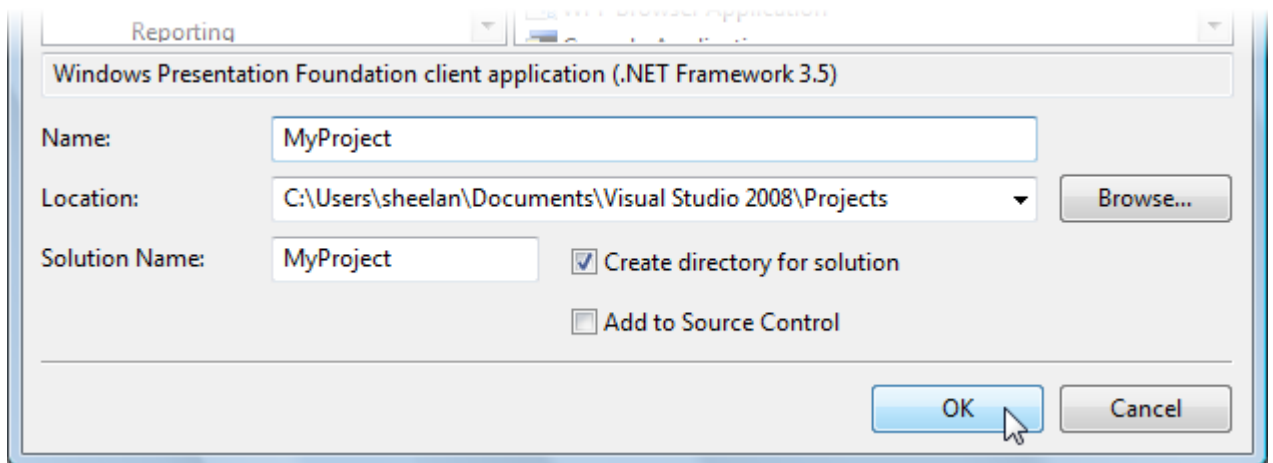
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

**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**.



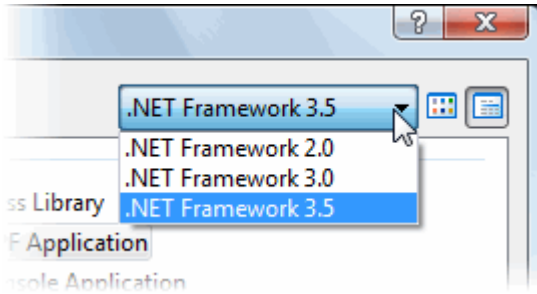
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

**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 ColorPicker for WPF Components to a Blend Project

In order to use **C1ColorPicker** or another **ComponentOne Studio for WPF** component in the Design workspace of Blend, you must first add references to the **C1.WPF**, **C1.WPF.Extended.dll**, and **WPFToolkit.dll** assemblies 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 **ColorPicker 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 references to the **C1.WPF**, **C1.WPF.Extended.dll**, and **WPFToolkit.dll** assemblies, 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 **C1ColorPicker**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1ColorPicker** component in the Toolbox to add it to **Window1.xaml**.

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

When you install **ComponentOne ColorPicker for WPF** the **C1ColorPicker** control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

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

- **C1ColorPicker**

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

### Manually Adding ColorPicker for WPF to the Toolbox

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

- C1ColorPicker

To manually add the C1ColorPicker 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 **ColorPicker for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1ColorPicker**, 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 ColorPicker for WPF to the Window

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

1. Add the C1ColorPicker control to the Visual Studio Toolbox.
2. Double-click C1ColorPicker or drag the control onto the window.

### Adding a Reference to the Assembly

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

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne ColorPicker 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 **ColorPicker for WPF** assembly visible to the project.

## Key Features

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

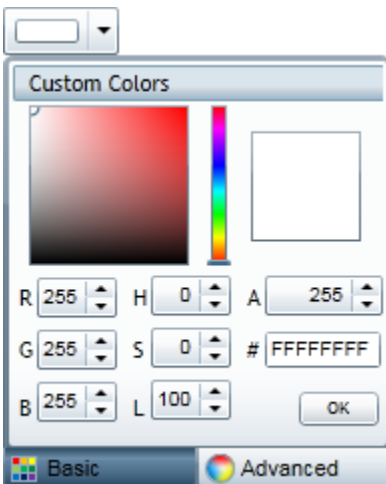
- **Select from 20+ Predefined Professionally Designed Palettes**

**ColorPicker** contains over 20 predefined color palettes that match the themes used in Microsoft Office. The colors in each palette go well together and can be used to create applications with a polished, professional appearance.



- **Built-in Color Editor for Custom Colors**

**ColorPicker** includes a color editor. This editor allows end-users to create colors that are not on the current palette using the RGB or HLS color models and including support for transparency.

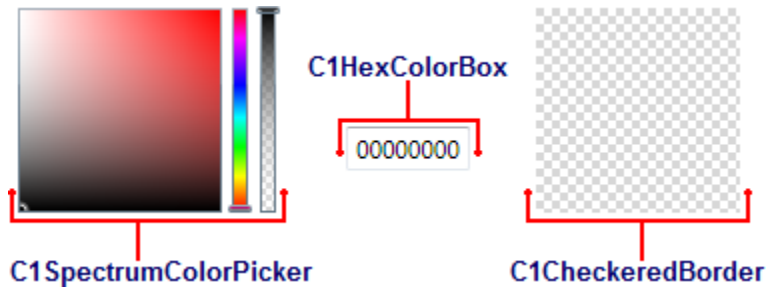


- **Different Views**

**C1ColorPicker** supports both simple and advanced views for color selection.

- **Composable Parts**

Each of the parts of the control can be used independently of the **C1ColorPicker** to create your custom controls. The **C1SpectrumColorPicker** control allows access to just the advanced color picking functionality of the **C1ColorPicker** control, the **C1HexColorBox** control provides data validation for hexadecimal code entries, and the **C1CheckedBorder** provides a simple way to display colors with transparencies.



- **Create Your Own Custom Palette**

If the available color palettes do not work for your application, you can create your own custom color palette. At run time, users can even be limited to selecting colors only from your chosen color palette.



# ColorPicker for WPF Quick Start

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

You will create a simple project using two **CIColorPicker** controls and a standard **Rectangle** control. The **CIColorPicker** controls will control a gradient that is applied to the **Rectangle**, so that choosing colors at run time will change the colors of the gradient – letting you explore the possibilities of using **ColorPicker for WPF**.

## Step 1 of 4: Setting up the Application

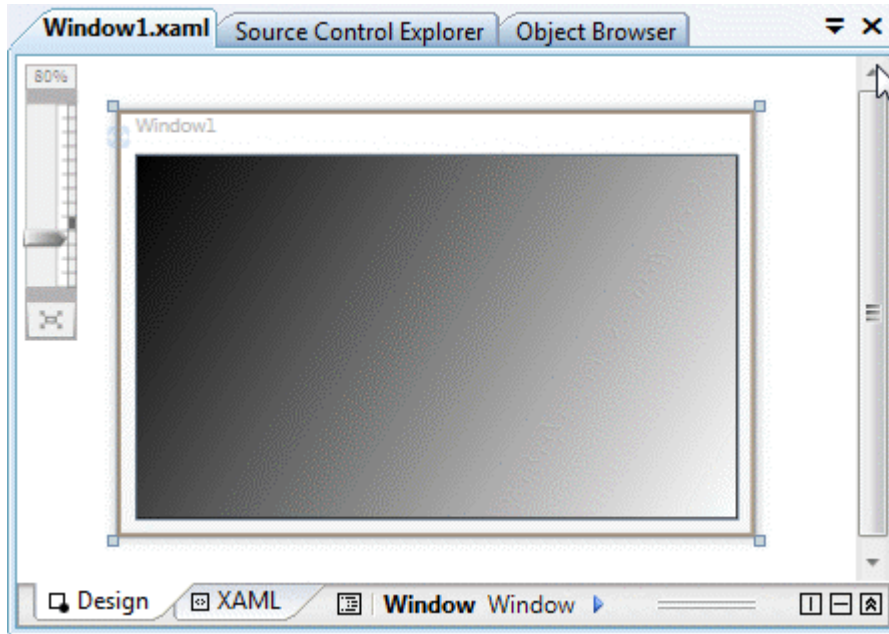
In this step you'll begin in Visual Studio to create a WPF application using **ColorPicker for WPF**. When you add a **CIColorPicker** control to your application, you'll have a complete, functional color input selector.

To set up your project and add a **CIColorPicker** control to your application, 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. Navigate to the Toolbox and double-click the **Rectangle** icon to add the standard control to the **Grid**.
3. Resize the window, and resize the rectangle to fill the window.
4. Switch to XAML view and add a **Fill** to the **<Rectangle>** tag so it appears similar to the following:

```
<Rectangle Name="Rectangle1" Stroke="Black">
  <Rectangle.Fill>
    <LinearGradientBrush x:Name="colors">
      <GradientStop x:Name="col1" Color="Black" Offset="0" />
      <GradientStop x:Name="col2" Color="White" Offset="1" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

This will add a black and white linear gradient fill to the rectangle. The design view of the page should now look similar to the following image:



You've successfully created a WPF application and customized the **Rectangle** control. In the next step you'll add and customize **C1ColorPicker** controls.

## Step 2 of 4: Adding C1ColorPicker Controls

In the previous step you created a new project and added a **Rectangle** control with a gradient to the application. In this step you'll continue by adding **C1ColorPicker** controls that will control the gradient fill in the **Rectangle**.

Complete the following steps:

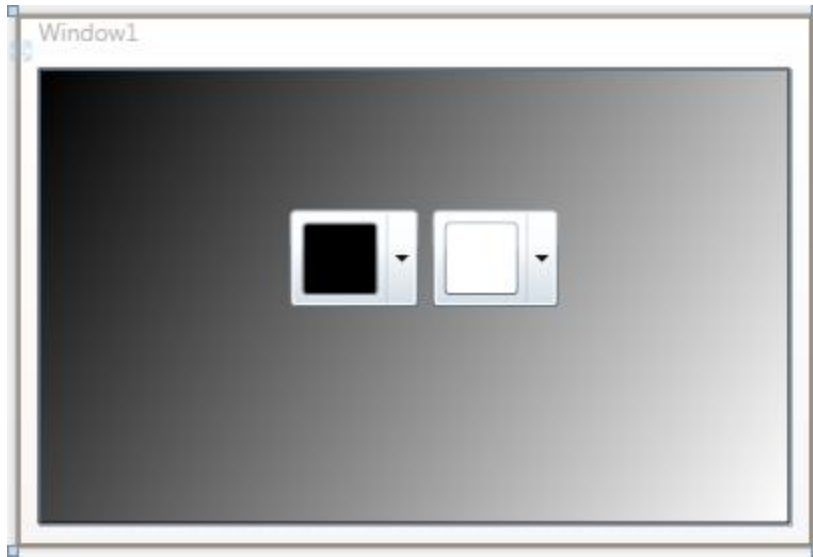
1. In Design view, click on the Rectangle to select it and navigate to the Visual Studio Toolbox.
2. In the Toolbox, locate and double-click the **C1ColorPicker** icon twice add two controls to the form.
3. Resize and position the two **C1ColorPicker** controls, so that they appear in the middle of the rectangle.
4. Click once on the first **C1ColorPicker** control, **C1ColorPicker1**, in Design view, navigate to the Properties window, and set the following properties:
  - Set **DropDownDirection** to **AboveOrBelow** to control how the control opens.
  - Set the **Mode** to **Advanced** so only the advanced color picker appears.
  - Set the **SelectedColor** to **Black** (or "#FF000000").

The XAML will appear similar to the following:

```
<c1ext:C1ColorPicker Margin="125,70,185,107.5" Name="C1ColorPicker1"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
DropDownDirection="AboveOrBelow" Mode="Advanced" SelectedBrush="Black" />
```

You'll leave the second **C1ColorPicker** control set to the default values.

The page's Design view should now look similar to the following image:



You've successfully set up your application's user interface, but if you run your application right now the color pickers will do nothing if you select a color. In the next step you'll add code to your application to add functionality to the controls.

## Step 3 of 4: Adding Code to the Application

In the previous steps you set up the application's user interface and added controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. Click once on the first **C1ColorPicker** control (**C1ColorPicker1**) to select it.
2. In the Properties window, click the lightning bolt **Events** icon to view control events.
3. Double-click in the text box next to the **SelectedColorChanged** event to switch to Code view and create the event handler.
4. In Code view, add the following import statements to the top of the page:

- Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

- C#

```
using C1.WPF;
using C1.WPF.Extended;
```

5. Add the following code just after the page's constructor to update the gradient values:

- Visual Basic

```
Private Sub UpdateGradient()
    If C1ColorPicker1 IsNot Nothing And C1ColorPicker2 IsNot Nothing
    Then
        Me.col1.Color = Me.C1ColorPicker1.SelectedColor
        Me.col2.Color = Me.C1ColorPicker2.SelectedColor
    End If
End Sub
```

- C#

```

void UpdateGradient()
{
    if (c1ColorPicker1 != null & c1ColorPicker2 != null)
    {
        this.col1.Color = this.c1ColorPicker1.SelectedColor;
        this.col2.Color = this.c1ColorPicker2.SelectedColor;
    }
}

```

6. Add code to the **C1ColorPicker1\_SelectedColorChanged** event handler so that it appears like the following:

- Visual Basic

```

Private Sub C1ColorPicker1_SelectedColorChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
C1ColorPicker1.SelectedColorChanged
    UpdateGradient()
End Sub

```

- C#

```

private void c1ColorPicker1_SelectedColorChanged(object sender,
C1.WPF.PropertyChangedEventArgs<System.Windows.Media.Color> e)
{
    UpdateGradient();
}

```

7. Return to Design view.
8. Click once on the second **C1ColorPicker** control (**C1ColorPicker2**) to select it.
9. In the Properties window, double-click in the text box next to the **SelectedColorChanged** event to switch to Code view and create the event handler (you may need to click the lightning bolt **Events** icon to view control events if events are not listed).
10. Add code to the **C1ColorPicker\_SelectedColorChanged** event handler so that it appears like the following:

- Visual Basic

```

Private Sub C1ColorPicker2_SelectedColorChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
c1cp1.SelectedColorChanged
    UpdateGradient()
End Sub

```

- C#

```

private void c1ColorPicker2_SelectedColorChanged(object sender,
C1.WPF.PropertyChangedEventArgs<System.Windows.Media.Color> e)
{
    UpdateGradient();
}

```

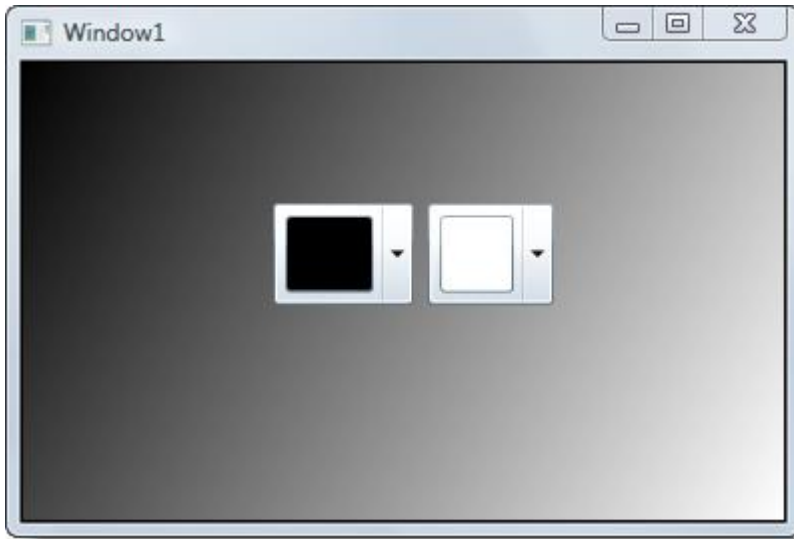
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

## Step 4 of 4: Running the Application

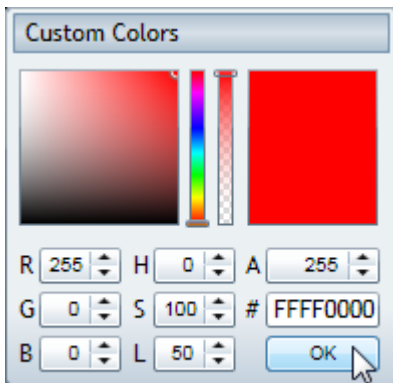
Now that you've created a WPF application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **ColorPicker for WPF's** run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

The application will appear similar to the following:

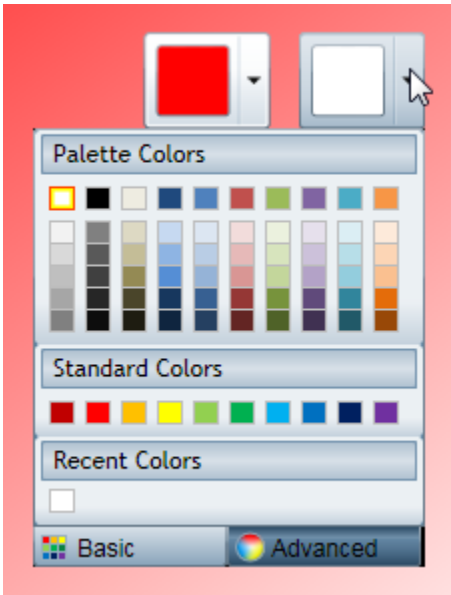


2. Click the drop-down arrow in the left color picker. Notice that the window opens above the drop-down box and that only the advanced mode is visible – this reflects the changes you made to the control. In advanced mode, users can specify any color and can use multiple methods of selecting a color.
3. Choose a color, for example **Red**, and click **OK**:



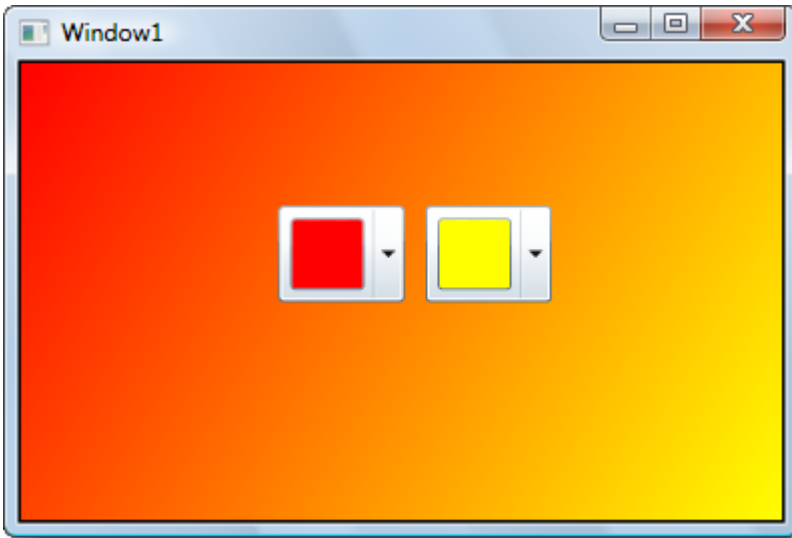
Notice that the control's selected color and the rectangle's gradient changes to reflect your color choice.

4. Click the drop-down arrow in the right color picker.



Notice that the **Basic** tab is visible (default). This tab displays **Palette Colors**, **Standard Colors**, and **Recent Colors**. You can pick any color and can also switch to the **Advanced** tab to pick a custom color. Note that the currently selected color is highlighted with a red border.

5. Pick a color, for example **Yellow**. The selected color will change and the background gradient of the rectangle will change to match your selection:



Congratulations! You've completed the **ColorPicker for WPF** quick start and created a simple WPF application, added and customized **ColorPicker for WPF** controls, and viewed some of the run-time capabilities of the controls.

# Working with ColorPicker for WPF

**ComponentOne ColorPicker for WPF** includes the `C1ColorPicker` control, a simple color selection control that lets users select colors from professionally designed palettes or your own custom colors. When you add the `C1ColorPicker` control to a XAML window, it exists as a complete color selection control that you can further customize.

The control's default interface looks similar to the following image:



For a description of the parts of the `C1ColorPicker` control, see the [Basic ColorPicker Mode](#) (page 26) and [Advanced ColorPicker Mode](#) (page 27) topics.

## Basic Properties

**ComponentOne ColorPicker for WPF** includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see [ColorPicker Appearance Properties](#) (page 33) for more information about properties that control appearance.

The following properties let you customize the `C1ColorPicker` control:

Property	Description
<code>DropDownDirection</code>	Specifies the expand direction of the control drop-down arrow.
<code>IsDropDownOpen</code>	Opens or closes the control drop-down box.
<code>Mode</code>	Indicates the mode of the color picker. Options include <b>Basic</b> , <b>Advanced</b> , and <b>Both</b> (default).
<code>Palette</code>	Gets/sets the palette to be used.
<code>SelectedBrush</code>	Gets the currently selected color as a <a href="#">Brush</a> .
<code>SelectedColor</code>	Gets/sets the currently selected color.
<code>ShowAlphaChannel</code>	Gets/sets whether the user can change the alpha channel (transparency value).

ShowRecentColors	Indicates if recently picked colors should be shown.
------------------	--

## Basic Events

**ComponentOne ColorPicker for WPF** includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1ColorPicker control:

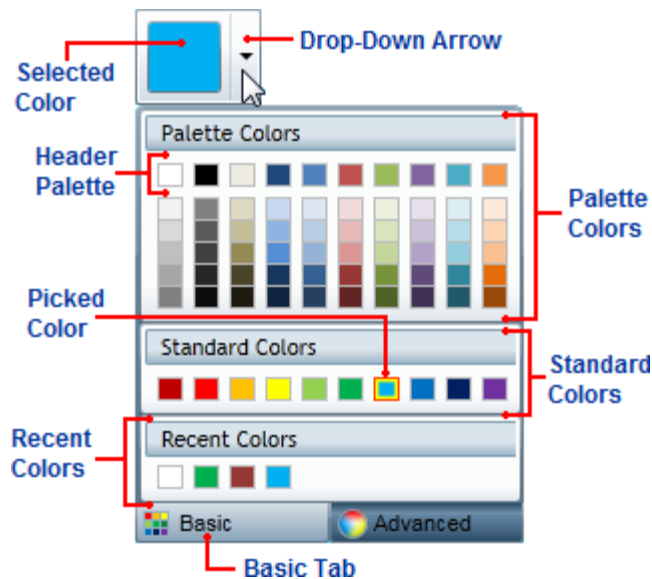
Event	Description
IsDropDownOpenChanged	Event raised when the IsDropDownOpen property has changed.
IsMouseOverChanged	Event raised when the <b>IsMouseOver</b> property has changed.
SelectedColorChanged	Event raised when the SelectedColor property has changed.

## ColorPicker Mode

The Mode property indicates if users should choose from a preselected palette of colors and/or if they can pick their own. Options include **Basic**, **Advanced**, and **Both**. By default, Mode is set to **Both** and both the **Basic** and **Advanced** tabs are visible. The following topics describe the two available tabs.

### Basic ColorPicker Mode

By default, the **C1ColorPicker** control will open with the **Basic** tab open when the control's drop-down arrow is clicked. The **Basic** tab appears similar to the following image:



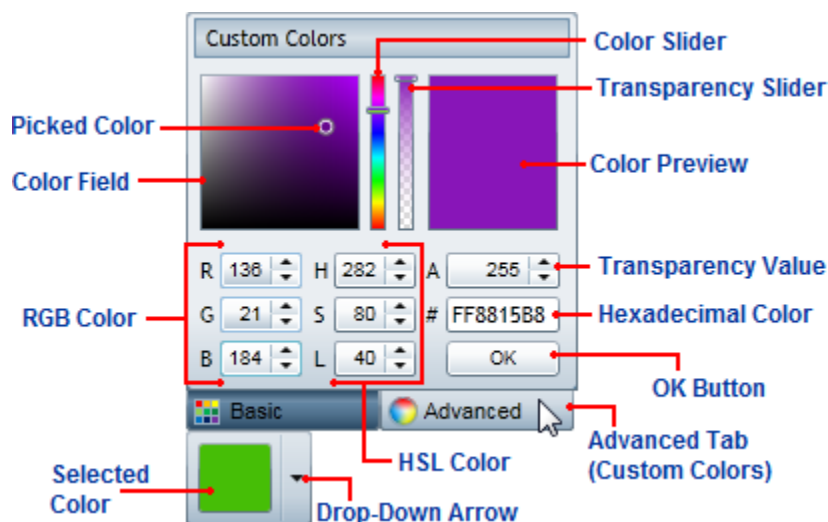
The **Basic** tab includes the following options/sections:

- **Drop-Down Arrow:** Click the drop-down arrow to open the C1ColorPicker control's window. See [Drop-Down Direction](#) (page 32) for information about setting where the drop-down window appears.
- **Basic Tab:** Click the **Basic** tab to access pre-selected colors at run time. Click the **Advanced** tab to choose a custom color. The Mode property must be set to **Basic** or **Both** for the **Basic** tab to be visible.

- **Selected Color:** The currently selected color will appear in the color picker's window.
- **Picked Color:** The currently picked color will appear with a red border in the list of colors.
- **Palette Colors:** Palette colors reflect the currently selected color palette. You can choose a palette by setting the Palette property.
- **Header Palette:** These colors are the basic colors of the palette – the expanded list of palette colors are typically variations of these basic colors.
- **Standard Colors:** Lists ten standard colors. These colors include a dark brick red, red, orange, yellow, light green, green, sky blue, blue, navy blue, and purple.
- **Recent Colors:** Lists up to ten recently selected colors. By default this section is visible, but you can choose to hide recent colors by setting the ShowRecentColors property to **False**. See [Recent Colors](#) (page 32) for more information.

## Advanced ColorPicker Mode

By default, the **CIColorPicker** control will open with the **Advanced** tab available when the control's drop-down arrow is clicked. The **Basic** tab appears by default, but the **Advanced** view can be selected by clicking the **Advanced** at the bottom of the control. The **Advanced** view appears similar to the following image:



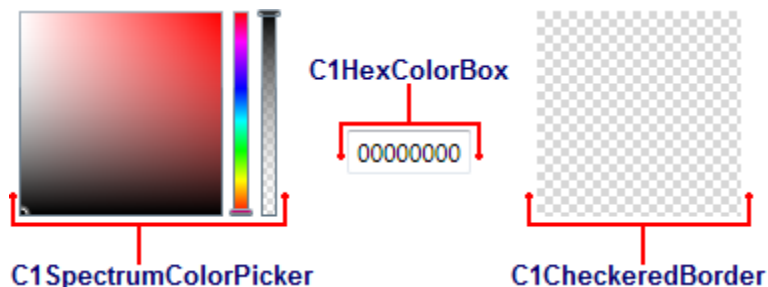
The **Advanced** tab includes the following options/sections:

- **Color Field/Picked Color:** The **Color Field** lets you choose a tone in a color's range. The **Picked Color** indicates the currently selected color. Move the **Color Slider** to pick a general color family and then fine tune the color selection in the **Color Field**.
- **Color Slider:** This slider lets you choose from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune the color selection in the **Color Field**.
- **Transparency Slider:** This slider lets you set the color's transparency. You can choose to make the color opaque or partially or completely transparent. Move the **Transparency Slider** to pick a transparency and note that the number in the **Transparency Value** box changes as well. This slider is only visible when the ShowAlphaChannel property is set to **True** (default)
- **Color Preview:** Preview the color you are currently choosing. Once you are satisfied with the color choice, click the **OK** button to close the drop-down box and set the color as the **Selected Color**.

- **Transparency Value:** This box lets you set the color's transparency. You can set the Transparency to a number between **0**, which is completely transparent, and **255**, which is completely opaque (default). When the ShowAlphaChannel property is set to **False** this box appears grayed out.
- **RGB Color:** These three numeric boxes let you choose a color using the Red Green Blue (RGB) color model.
- **HSL Color:** These three numeric boxes let you choose a color using the Hue Saturation Lightness (HSL) color model.
- **Hexadecimal Color:** If eight digits are visible, the first two digits represent the color's transparency ranging from FF (opaque) to 00 (transparent) and the last six digits represent standard hexadecimal color selection. Note that if the ShowAlphaChannel property is set to **False**, only the last six digits will be visible (no transparency value). For more information about hexadecimal color selection, see [w3schools](http://w3schools.com).
- **OK Button:** Once you are satisfied with the color choice, click the **OK** button to close the drop-down box and set the color as the **Selected Color**.
- **Selected Color:** The currently selected color will appear in the color picker's window.
- **Drop-Down Arrow:** Click the drop-down arrow to open the C1ColorPicker control's window. See [Drop-Down Direction](#) (page 32) for information about setting where the drop-down window appears.
- **Advanced Tab:** Click the **Advanced** tab to choose a custom color at run time. Click the **Basic** tab to view pre-selected colors. The Mode property must be set to **Advanced** or **Both** for the **Advanced** tab to be visible.

## Additional Controls

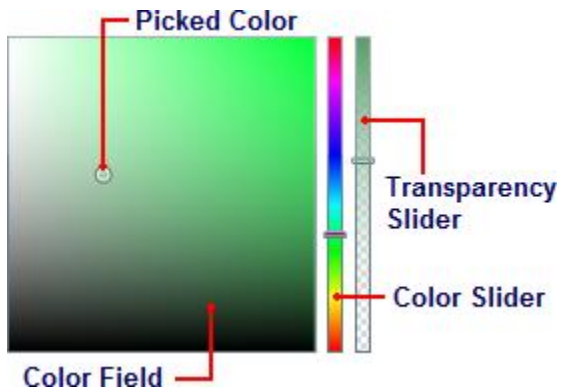
In addition to the full-featured C1ColorPicker control, **ColorPicker for WPF** includes parts of the C1ColorPicker control, that allow you to customize color picking in your application:



The [C1SpectrumColorPicker](#) control allows access to just the advanced color picking functionality of the C1ColorPicker control, the [C1HexColorBox](#) control provides data validation for hexadecimal code entries, and the [C1CheckedRedBorder](#) provides a simple way to display colors with transparencies. The following topics describe these parts.

### C1SpectrumColorPicker

The C1SpectrumColorPicker control allows access to just the advanced color picking functionality of the C1ColorPicker control. The C1SpectrumColorPicker control appears similar to the following image:



The C1SpectrumColorPicker control includes the following options/sections:

- **Color Field/Picked Color:** The **Color Field** lets you choose a tone in a color's range. The **Picked Color** indicates the currently selected color. Move the **Color Slider** to pick a general color family and then fine tune the color selection in the **Color Field**.
- **Color Slider:** This slider lets you choose from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune the color selection in the **Color Field**.
- **Transparency Slider:** This slider lets you set the color's transparency. You can choose to make the color opaque or partially or completely transparent. Move the **Transparency Slider** to pick a transparency. This slider is only visible when the ShowAlphaChannel property is set to **True** (default)

## C1HexColorBox

The C1HexColorBox control provides data validation for hexadecimal code entries. For example the basic C1HexColorBox control appears similar to the following image:

A0FFF700

The C1HexColorBox control appears similar to a regular text box. By default, the C1HexColorBox control appears with eight digits. If eight digits are visible, the first two digits represent the color's transparency ranging from FF (opaque) to 00 (transparent) and the last six digits represent standard hexadecimal color selection. For more information about hexadecimal color selection, see [w3schools](http://w3schools.com).

Note that if the ShowAlphaChannel property is set to **False**, only the last six digits will be visible (no transparency value will be included):

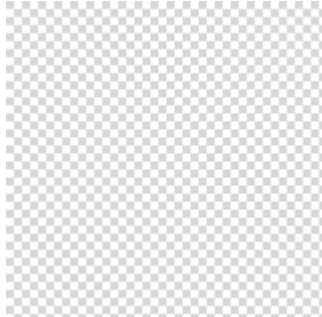
D45656

You can also, if you choose, choose to display a '#' symbol to the start of the C1HexColorBox control by setting the ShowSharpPrefix property to **True**:

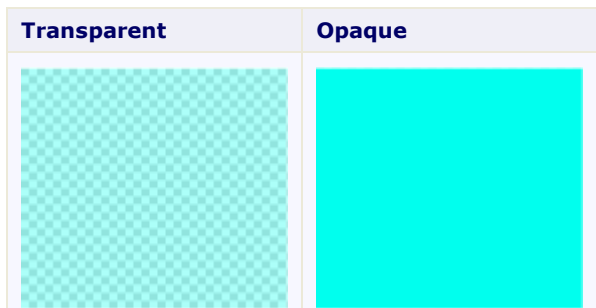
#0F00C48E

## C1CheckedBorder

The C1CheckedBorder control provides a simple way to display colors with set alpha values so you can display colors with varying transparencies. By default, the control appears similar to the following image:



The C1CheckedBorder control supports both transparent and opaque color values:



## Available ColorPicker Palettes

**ColorPicker for WPF** includes over 20 predefined color palettes that match the themes used in Microsoft Office. The colors in each palette go well together and can be used to create applications with a polished, professional appearance. To change the color palette, you can set the Palette property. For more information, see [Setting the Palette](#) (page 41).

The following built-in palettes are available:

Name	Palette
Apex	
Aspect	

Name	Palette
Office	
Opulent	

Civic	
Concourse	
Default	
Equity	
Flow	
Foundry	
GrayScale	
Median	
Metro	
Module	

Oriel	
Origin	
Paper	
Solstice	
Standard	
Technic	
Trek	
Urban	
Verve	

## Recent Colors

By default, when the user views the C1ColorPicker control's **Basic** tab at run time, along with the selected color palette and the standard color palette, the tab includes a section that lists recently picked colors:



If you choose, you can turn the display of recent colors off. The ShowRecentColors property sets whether or not these colors are displayed. For more information and an example, see [Hiding Recent Colors](#) (page 46).

## Drop-Down Direction

By default, when the user clicks the C1ColorPicker control's drop-down arrow at run-time the color picker will appear below the control, and, if that is not possible, above the control. However, you can customize where you would like the color picker to appear by setting the DropDownDirection property.

You can set the DropDownDirection property to one of the following options:

Event	Description
BelowOrAbove (default)	Tries to open the drop-down <b>C1ComboBox</b> below the header. If it is not possible tries to open above it.
AboveOrBelow	Tries to open the drop-down <b>C1ComboBox</b> above the header. If it is not possible tries to open below it.
ForceBelow	Forces the drop-down <b>C1ComboBox</b> to open below the header.
ForceAbove	Forces the <b>C1ComboBox</b> content to open above the header.

For more information and an example, see [Changing the Drop-Down Window Direction](#) (page 45).

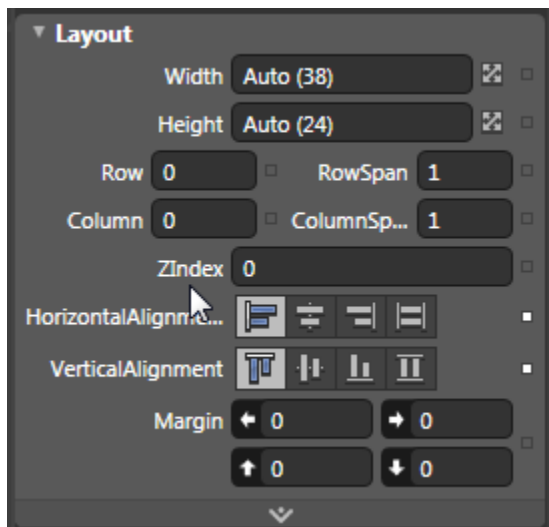
# ColorPicker Layout and Appearance

The following topics detail how to customize the C1ColorPicker 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 lay out the control and to customize the control's actions.

## Layout in a Panel

You can easily lay out the `C1ColorPicker` and other controls in your WPF application, using the attached layout properties. For example, you can lay out your control in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties and in a **Canvas** panel with its **Left** and **Top** properties. For example, the `C1ColorPicker` control includes the following **Layout** properties when located within a **Grid** panel:



You can change the sizing, alignment, and location of the `C1ColorPicker` control within the **Grid** panel.

## ColorPicker Appearance Properties

**ComponentOne ColorPicker for WPF** includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

### Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
<a href="#">Background</a>	Gets or sets a brush that describes the background of a control. This is a dependency property.
<a href="#">Foreground</a>	Gets or sets a brush that describes the foreground color. This is a dependency property.

### Alignment Properties

The following properties let you customize the control's alignment:

Property	Description
<a href="#">HorizontalAlignment</a>	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property.
<a href="#">VerticalAlignment</a>	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.

## Border Properties

The following properties let you customize the control's border:

Property	Description
<a href="#">BorderBrush</a>	Gets or sets a brush that describes the border background of a control. This is a dependency property.
<a href="#">BorderThickness</a>	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 **CIColorPicker** control:

Property	Description
<a href="#">Height</a>	Gets or sets the suggested height of the element. This is a dependency property.
<a href="#">MaxHeight</a>	Gets or sets the maximum height constraint of the element. This is a dependency property.
<a href="#">MaxWidth</a>	Gets or sets the maximum width constraint of the element. This is a dependency property.
<a href="#">MinHeight</a>	Gets or sets the minimum height constraint of the element. This is a dependency property.
<a href="#">MinWidth</a>	Gets or sets the minimum width constraint of the element. This is a dependency property.
<a href="#">Width</a>	Gets or sets the width of the element. This is a dependency property.

## 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.

## ClearStyle Properties

The following table lists all of the ClearStyle-supported properties in the C1ColorPicker control as well as a description of the property:

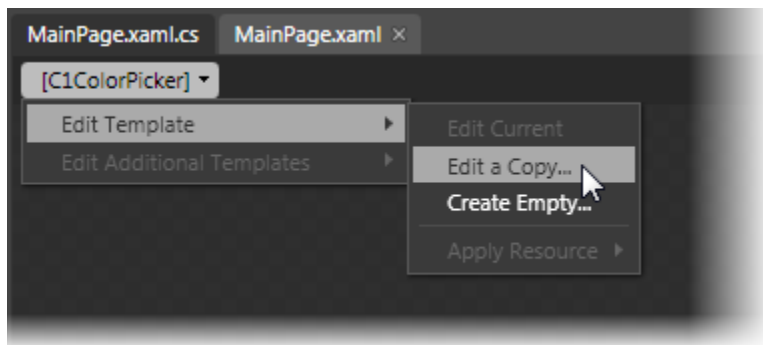
Property	Description
<b>Background</b>	Gets or sets a brush that describes the background of a control. The default <b>Background</b> color is LightBlue.
FocusBrush	A brush used to define the appearance of the control, when the control is in focus.
MouseOverBrush	A brush used to define the appearance of the control, when the control is in moused over.
PressedBrush	A brush used to define the appearance of the control, when the control is selected.

## ColorPicker 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 ColorPicker 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 C1ColorPicker control and, in the menu, selecting **Edit Control Parts (Templates)**. 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.

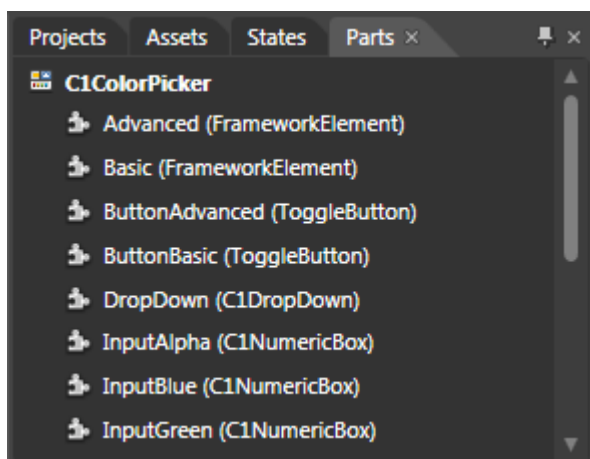
## ColorPicker Styles

**ComponentOne ColorPicker for WPF's C1ColorPicker** control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
ColorContainerStyle	Sets/gets the style of the <b>ItemsControl</b> used to show a section of colors (that is recent colors).
<a href="#">FontStyle</a>	Gets or sets the font style. This is a dependency property.
<a href="#">Style</a>	Gets or sets the style used by this element when it is rendered. This is a dependency property.

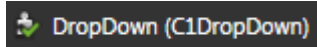
## ColorPicker Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1ColorPicker** control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

In the **Parts** window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** window will change to indicate selection:



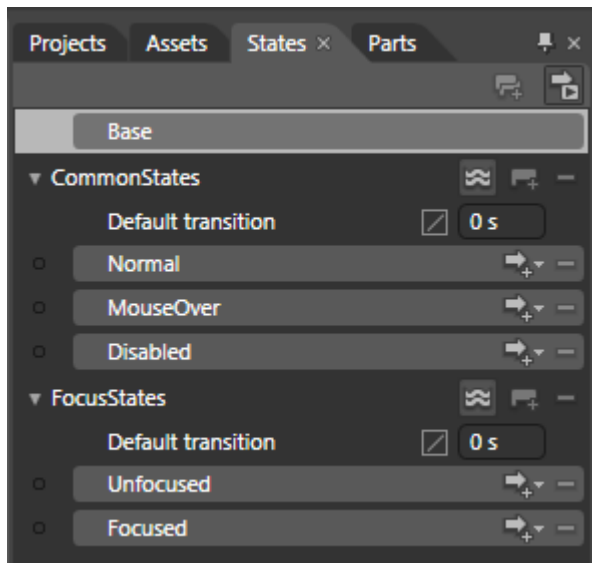
Template parts available in the **C1ColorPicker** control include:

Name	Type	Description
Advanced	<a href="#">FrameworkElement</a>	Provides a framework of common APIs for objects that participate in WPF layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in WPF.
Basic	<a href="#">FrameworkElement</a>	Provides a framework of common APIs for objects that participate in WPF layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in WPF.
ButtonAdvanced	<a href="#">ToggleButton</a>	Base class for controls that can switch states, such as <a href="#">CheckBox</a> and <a href="#">RadioButton</a> .
ButtonBasic	<a href="#">ToggleButton</a>	Base class for controls that can switch states, such as <a href="#">CheckBox</a> and <a href="#">RadioButton</a> .
DropDown	<a href="#">Grid</a>	Defines a flexible grid area that consists of columns and rows.
InputAlpha	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputBlue	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputGreen	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputHue	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputLuminance	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputRed	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputSaturation	<b>C1NumericBox</b>	The <b>C1NumericBox</b> control is a numeric editor that allows you to display and edit numeric values in many formats.
InputWeb	<a href="#">TextBox</a>	Represents a control that can be used to display single-format, multi-line text.
OkButton	<a href="#">Button</a>	Represents a button control.
Preview	<a href="#">Rectangle</a>	Draws a rectangle shape, which can have a stroke

		and a fill.
RecentColors	<a href="#">Grid</a>	Defines a flexible grid area that consists of columns and rows.
RecentColorsHeader	<a href="#">FrameworkElement</a>	Provides a framework of common APIs for objects that participate in WPF layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in WPF.
Root	<a href="#">FrameworkElement</a>	Provides a framework of common APIs for objects that participate in WPF layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in WPF.
Spectrum	<a href="#">C1SpectrumColorPicker</a>	Represents a sliding color picker.
StandardColors	<a href="#">Grid</a>	Defines a flexible grid area that consists of columns and rows.
ThemeColorsHeader	<a href="#">Grid</a>	Defines a flexible grid area that consists of columns and rows.
ThemeColorsValues	<a href="#">Grid</a>	Defines a flexible grid area that consists of columns and rows.

## ColorPicker Visual States

In Microsoft Expression Blend, you can add custom states and state groups to define a different appearance for each state of your user control – for example, the visual state of the control could change on mouse over. You can view and edit visual states by creating a new template and [adding a new template part](#) (page 36). Once you've done so the available visual states for that part will be visible in the **Visual States** window:



Common states include **Normal** for the normal appearance of the item, **MouseOver** for the item on mouse over, and **Disabled** for when the item is not enabled. Focus states include **Unfocused** for when the item is not in focus and **Focused** when the item is in focus.

## XAML Elements

Several auxiliary XAML elements are installed with **ComponentOne ColorPicker for WPF**. These elements include templates and themes and are located in the **ColorPicker for WPF** installation directory. You can incorporate these elements into your project, for example, to create your own theme based on the default theme.

### Included Auxiliary XAML Elements

The following auxiliary XAML element is included with **ColorPicker for WPF**:

Element	Folder	Description
generic.xaml	XAML	Specifies the templates for different styles and the initial style of the control.



# ColorPicker 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>ColorPicker</b> page in the <b>ControlExplorer</b> sample demonstrates how to customize the C1ColorPicker control.

# ColorPicker 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 C1ColorPicker control in general. If you are unfamiliar with the **ComponentOne ColorPicker for WPF** product, please see the [ColorPicker for WPF Quick Start](#) (page 19) first.

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

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

## Setting the Palette

**ColorPicker for WPF** includes over 20 predefined color palettes that match the themes used in Microsoft Office. For more information about palette choices, see [Available ColorPicker Palettes](#) (page 30). To change the color palette, you can set the Palette property.

To set the Palette property, complete the following steps:

1. Navigate to the Toolbox and double-click the Button icon to add the control to the project.
2. Resize and reposition the Button on the form.
3. Navigate to the Properties window and set the button's **Content** property to "Change Palette".
4. Double-click the button to switch to Code view and create the **Button\_Click** event handler.
5. Add code for the **Button\_Click** event handler, so it appears like the following:

- Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    ' Set the color palette.
    Me.C1ColorPicker.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale)
End Sub
```

- C#

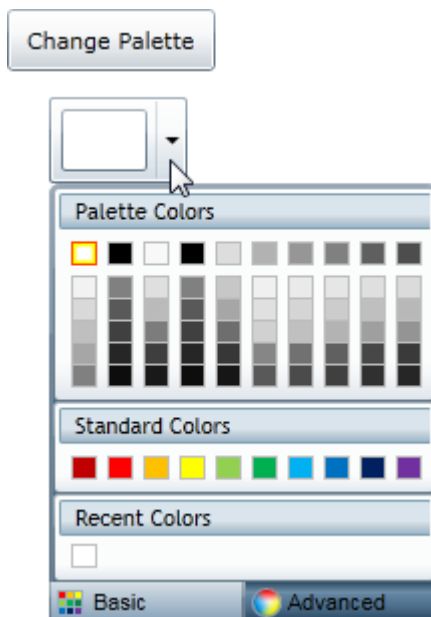
```
private void button1_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    // Change color palette.
    this.c1ColorPicker.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale);
}
```

The **C1ColorPicker**'s color palette will now change to grayscale when the button is clicked.

### Run the application and observe:

Observe the following:

1. Click the C1ColorPicker control's drop-down arrow and notice that the default palette appears.
2. Click the **Change Palette** button and click the C1ColorPicker control's drop-down arrow once again. Notice that a grayscale palette appears:



## Creating a Custom Palette

**ColorPicker for WPF** includes over 20 predefined color palettes that match the themes used in Microsoft Office, but if you choose you can create your own custom color palette rather than using a predefined one. In the following steps you'll create a custom palette, and when a button is pressed, apply that palette to the C1ColorPicker control.

To create a custom palette, complete the following steps:

1. Navigate to the Toolbox and double-click the Button icon to add the control to the project.
2. Resize and reposition the Button on the form.
3. Navigate to the Properties window and set the button's **Content** property to "Change Palette".
4. Double-click the button to switch to Code view and create the **Button\_Click** event handler.
5. Add code for the **Button\_Click** event handler, so it appears like the following:
  - Visual Basic

```

Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    ' Set the color palette.
    Dim cp1 as New ColorPalette("Pittsburgh")
    cp1.Clear()
    cp1.Add(Color.FromArgb(255, 0, 0, 0))
    cp1.Add(Color.FromArgb(255, 99, 107, 112))
    cp1.Add(Color.FromArgb(255, 255, 255, 255))
    cp1.Add(Color.FromArgb(255, 247, 181, 18))
    cp1.Add(Color.FromArgb(255, 253, 200, 47))
    cp1.Add(Color.FromArgb(255, 43, 41, 38))
    cp1.Add(Color.FromArgb(255, 149, 123, 77))
    cp1.Add(Color.FromArgb(255, 209, 201, 157))
    cp1.Add(Color.FromArgb(255, 0, 33, 71))
    cp1.Add(Color.FromArgb(255, 99, 177, 229))
    c1ColorPicker1.Palette = cp1
End Sub

```

- **C#**

```

private void button1_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    // Set the color palette.
    ColorPalette cp1 = new ColorPalette("Pittsburgh");
    cp1.Clear();
    cp1.Add(Color.FromArgb(255, 0, 0, 0));
    cp1.Add(Color.FromArgb(255, 99, 107, 112));
    cp1.Add(Color.FromArgb(255, 255, 255, 255));
    cp1.Add(Color.FromArgb(255, 247, 181, 18));
    cp1.Add(Color.FromArgb(255, 253, 200, 47));
    cp1.Add(Color.FromArgb(255, 43, 41, 38));
    cp1.Add(Color.FromArgb(255, 149, 123, 77));
    cp1.Add(Color.FromArgb(255, 209, 201, 157));
    cp1.Add(Color.FromArgb(255, 0, 33, 71));
    cp1.Add(Color.FromArgb(255, 99, 177, 229));
    c1ColorPicker1.Palette = cp1;
}

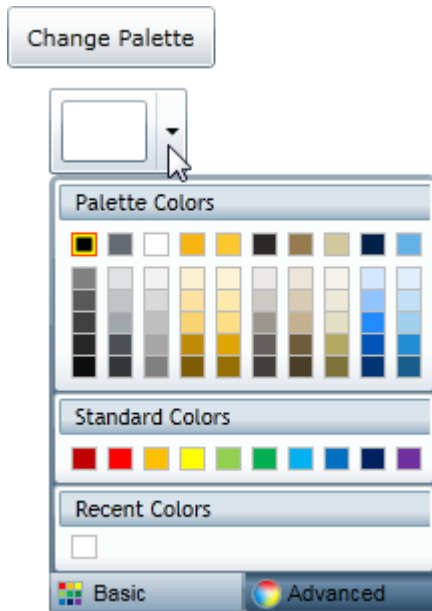
```

The **ColorPicker**'s color palette will now change to a custom palette when the button is clicked.

**Run the application and observe:**

Observe the following:

1. Click the C1ColorPicker control's drop-down arrow, and notice that the default palette appears.
2. Click the **Change Palette** button and click the C1ColorPicker control's drop-down arrow once again. Notice that the custom palette appears:



## Changing the Background Color

The **Background** property gets or sets the value of the C1ColorPicker control's background color. By default the C1ColorPicker control starts with the **Background** property unset, but you can customize this at design time, in XAML, and in code.

### At Design Time in Blend

To set the **Background** property at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the Properties window, and click the **Background** drop-down arrow, and choose **Red** or another color in the color picker.

### In XAML

For example, to set the **Background** property to **Red** add `Background="Red"` to the `<c1: C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" Background="Red"/>
```

### In Code

For example, to set the **Background** property to **Red**, add the following code to your project:

- Visual Basic

```
Me.C1ColorPicker1.Background = System.Windows.Media.Brushes.Red
```
- C#

```
this.c1ColorPicker1.Background = System.Windows.Media.Brushes.Red;
```

### Run the application and observe:

The background of the C1ColorPicker control will appear red:



## Changing the Drop-Down Window Direction

By default, when the user clicks the C1ColorPicker control's drop-down arrow at run-time the color picker will appear below the control, and if that is not possible, above the control. However, you can customize where you would like the color picker to appear. For more information about the drop-down arrow direction, see [Drop-Down Direction](#) (page 32).

### At Design Time in Blend

To change the drop-down window direction at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the Properties window and click the DropDownDirection drop-down arrow.
3. Choose an option, for example **ForceAbove**.

This will set the DropDownDirection property to the option you chose.

### In XAML

For example, change the drop-down window direction add `DropDownDirection="ForceAbove"` to the `<c1:C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45" VerticalAlignment="Top" DropDownDirection="ForceAbove"/>
```

### In Code

For example, to change the drop-down window direction, add the following code to your project:

- Visual Basic  

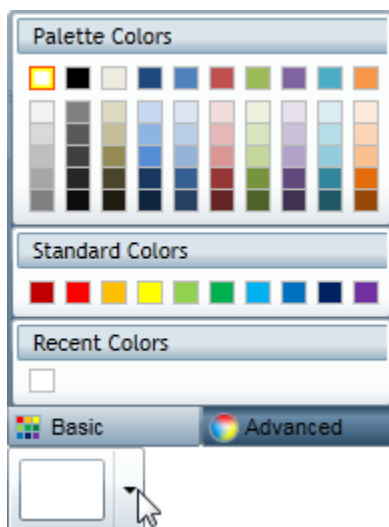
```
Me.C1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove
```
- C#  

```
this.c1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove;
```

This will set the DropDownDirection property to **ForceAbove**.

### Run the application and observe:

When you click the C1ColorPicker control's drop-down arrow, the drop down window will appear above the control:



## Hiding Recent Colors

By default the C1ColorPicker control displays recent colors in the **Basic** tab of the color picker window. For more information, see [Recent Colors](#) (page 32). If you choose, you prevent recent colors from being displayed at design time, in XAML, and in code.

### At Design Time in Blend

To prevent recent colors from being displayed at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the Properties window.
3. Locate the **ShowRecentColors** property and set it to **False**.

### In XAML

To prevent recent colors from being displayed add `ShowRecentColors="False"` to the `<c1:C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
  VerticalAlignment="Top" ShowRecentColors="False" />
```

### In Code

To prevent recent colors from being displayed, add the following code to your project:

- Visual Basic  

```
Me.C1ColorPicker1.ShowRecentColors = False
```
- C#  

```
this.c1ColorPicker1.ShowRecentColors = false;
```

### Run the application and observe:

When you click the C1ColorPicker control's drop-down arrow, observe that recent colors are not displayed on the Basic tab:

