
ComponentOne

DateTimePicker for WPF

Copyright © 1987-2010 ComponentOne LLC. All rights reserved.

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com
Web site: <http://www.componentone.com>

Sales

E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

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

Warranty

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

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

Copying and Distribution

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

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

DateTimePicker for WPF	1
Installing DateTimePicker for WPF	1
DateTimePicker for WPF Setup Files	1
System Requirements	2
Installing Demonstration Versions	2
Uninstalling DateTimePicker for WPF	2
End-User License Agreement	3
Licensing FAQs	3
What is Licensing?	3
How does Licensing Work?	3
Common Scenarios	4
Troubleshooting	6
Technical Support	7
Redistributable Files	8
About this Documentation	8
XAML and XAML Namespaces	9
Creating a Microsoft Blend Project	9
Creating a .NET Project in Visual Studio	10
Creating an XAML Browser Application (XBAP) in Visual Studio	11
Adding the DateTimePicker for WPF Components to a Blend Project	12
Adding the DateTimePicker for WPF Components to a Visual Studio Project	13
Key Features.....	15
DateTimePicker for WPF Quick Start	15
Step 1 of 3: Creating an Application with a C1DateTimePicker Control.....	15
Step 2 of 3: Customizing the Control	15
Step 3 of 3: Running the Project.....	16
Working with C1DateTimePicker	17
C1DateTimePicker Elements.....	17
Edit Modes.....	18
Date Format.....	18
Time Format	18
DateTimePicker for WPF Layout and Appearance	19
DateTimePicker for WPF Appearance Properties	19
Text Properties	19
Color Properties	19
Border Properties.....	19
Size Properties.....	20
Templates.....	20
DateTimePicker for WPF Task-Based Help	23
Allowing Null Values	23
Selecting the Edit Mode.....	24
Selecting the Time Format.....	24
Selecting the Date Format	25
Setting the Minimum and Maximum Calendar Dates	26
Specifying the Date and Time.....	28

DateTimePicker for WPF

Exchange date and time information using **ComponentOne DateTimePicker™ for Silverlight**. It provides a simple and intuitive UI for selecting date and time or just time values. Users can select the date and time using the spin buttons, keyboard arrows, or by typing in fields.

Getting Started

- [Working with C1DateTimePicker](#) (page 17)
- [Quick Start](#) (page 15)
- [Task-Based Help](#) (page 23)

Installing DateTimePicker for WPF

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

DateTimePicker 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 DateTimePicker for WPF**, the following DLLs are installed:

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

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

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

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

H2Help Contains online documentation for all Studio components.

C1WPF\XAML Contains the full XAML definitions of C1DateTimePicker styles and templates which can be used for creating your own custom styles and

templates.

Samples

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

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

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

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

Common	Contains support and data files that are used by many of the demo programs.
C1WPF	Contains samples for DateTimePicker for WPF .

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2) Windows Vista™ 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:	DateTimePicker 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 **DateTimePicker 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 DateTimePicker 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 DateTimePicker for WPF

To uninstall **ComponentOne DateTimePicker for WPF**:

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

2. Select **ComponentOne Studio for WPF** and click the **Remove** button.
3. Click **Yes** to remove the program.

End-User License Agreement

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

Licensing FAQs

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

What is Licensing?

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

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

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

How does Licensing Work?

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

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

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

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

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

These files are automatically added to the project.

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

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

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

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

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

Common Scenarios

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

Creating components at design time

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

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

Creating components at run time

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

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

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

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

Inheriting from licensed components

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

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

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

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


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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

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

Using licensed components with automated testing products

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

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

For example:

```
#if AUTOMATED_TESTING
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")] ]
```

```
#endif
public class MyDerivedControl : C1LicensedControl
{
    // ...
}
```

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

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

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

Troubleshooting

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

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

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

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

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

If that fails follow these steps:

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

Alternatively, follow these steps:

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

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

1. Open the project and go to the Solution Explorer window.
2. Find the **licenses.licx** file and right-click it.

3. Select the **Rebuild Licenses** option (this will rebuild the **App_Licenses.licx** file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Support via [HelpCentral](#)**
ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of [FAQs](#), [samples](#), [Version Release History](#), [Articles](#), searchable [Knowledge Base](#), searchable [Online Help](#) and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
ComponentOne peer-to-peer product [forums and newsgroups](#) are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for

users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**

Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end users in an application.

- **Documentation**

ComponentOne documentation is installed with each of our products and is also available online at [HelpCentral](#). 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 DateTimePicker 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.DateTimePicker.dll
- C1.WPF.DateTimePicker.Expression.Design.dll
- C1.WPF.DateTimePicker.VisualStudio.Design.dll

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

- WPFToolkit.dll

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

About this Documentation

You can create your applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

Acknowledgements

Microsoft, Windows, Windows Vista, Windows 7, Visual Studio, and Microsoft Expression are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

ComponentOne

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

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

412.681.4343

412.681.4384 (Fax)

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

ComponentOne Doc-To-Help

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

XAML and XAML Namespaces

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

XAML Namespaces

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

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

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

When you add a `C1DateTimePicker` 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:my="clr-namespace:C1.WPF.DateTimePicker;assembly=C1.WPF.DateTimePicker"
```

The namespace value is **my** and the namespace is **C1.WPF.DateTimePicker**.

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

```
xmlns:MyMTB=http://schemas.componentone.com/wpf/C1DateTimePicker
```

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

```
<MyMTB:C1DateTimePicker Name="c1DateTimePicker1"
    BorderThickness="10,10,10,10">
```

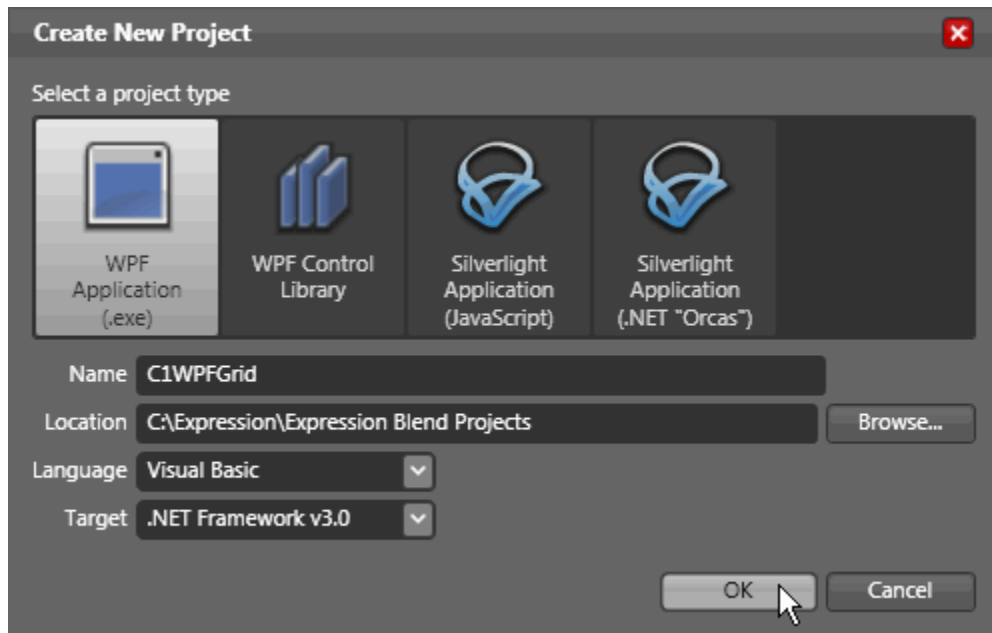
Creating a Microsoft Blend Project

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**.

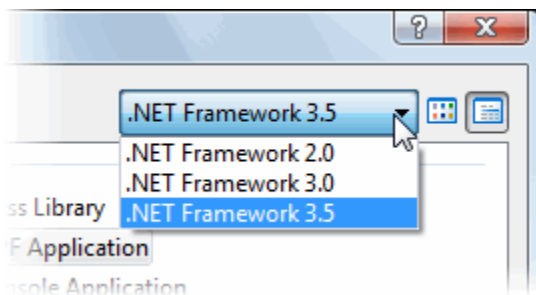


A new Blend project with a XAML window is created.

Creating a .NET Project in Visual Studio

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

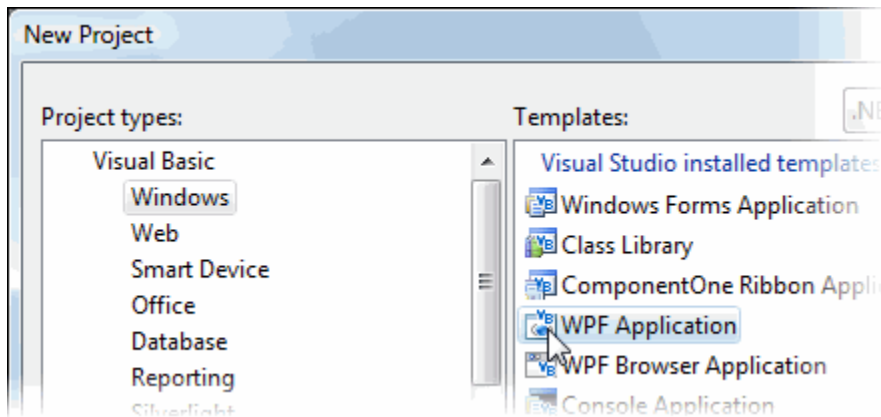
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**. The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



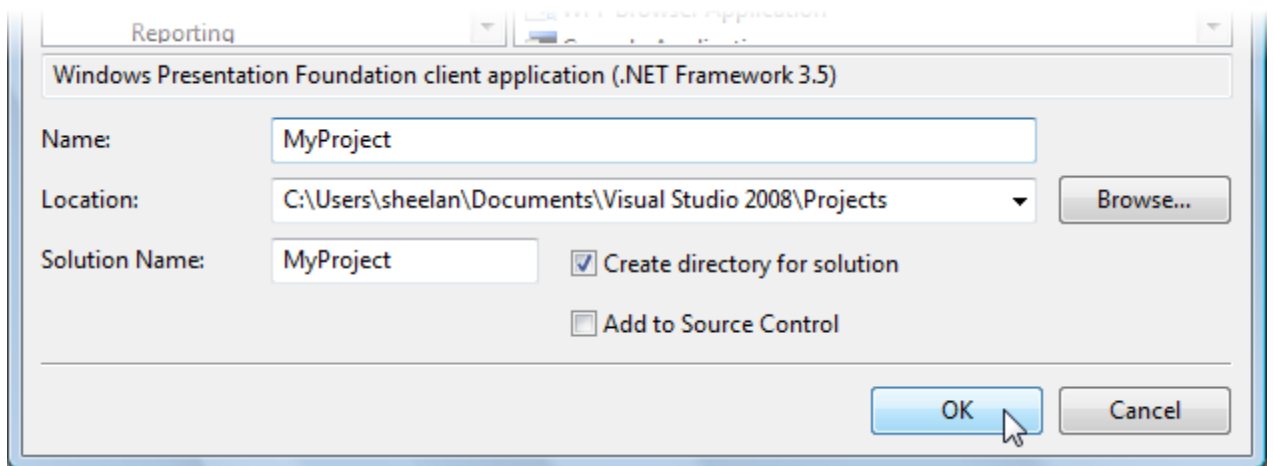
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

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.

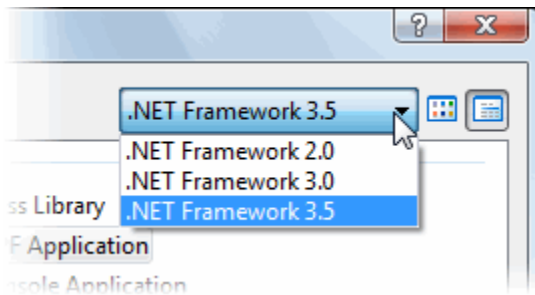
Note: You can create your WPF applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, Blend will be used for most examples.

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 DateTimePicker for WPF Components to a Blend Project

In order to use `C1DateTimePicker` or another **ComponentOne DateTimePicker for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF.Extended** assembly and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

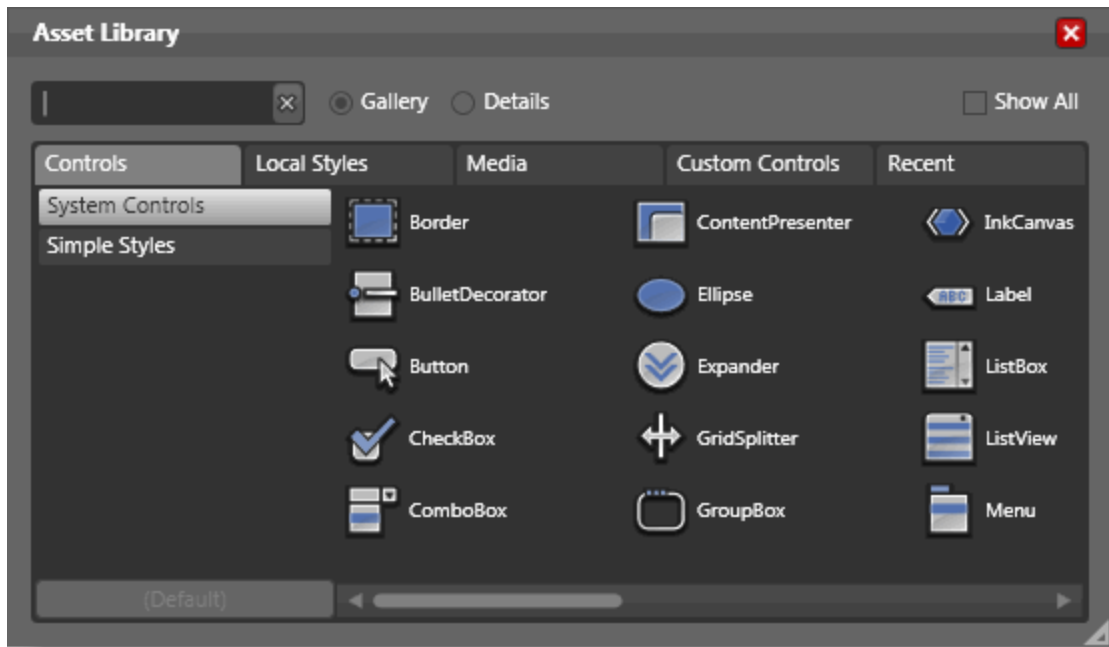
1. Select **Project | Add Reference**.
2. Browse to find the **C1.WPF.DateTimePicker.dll** assembly installed with **DateTimePicker for WPF**.

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

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



2. Click the **Custom Controls** tab. All of the **DateTimePicker for WPF** main and auxiliary components are listed here.
3. Select **C1DateTimePicker**. The component will appear in the Toolbox above the **Asset Library** button.
4. Double-click the **C1DateTimePicker** component in the Toolbox to add it to **Window1.xaml**.

Adding the DateTimePicker for WPF Components to a Visual Studio Project

When you install **ComponentOne DateTimePicker for WPF** the C1DateTimePicker control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne DateTimePicker for WPF provides the following control:

- C1DateTimePicker

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

Manually Adding DateTimePicker for WPF to the Toolbox

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

- C1DateTimePicker

To manually add the C1DateTimePicker 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 **DateTimePicker for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFDateTimePicker**, 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.DateTimePicker** namespace. Note that there may be more than one component for each namespace.

Adding DateTimePicker for WPF to the Window

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

1. Add the C1DateTimePicker control to the Visual Studio Toolbox.
2. Double-click C1DateTimePicker or drag the control onto the window.

Adding a Reference to the Assembly

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

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

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

Key Features

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

- **Select from Several Displayable Versions**

Choose one of the available edit modes: **DateTime** (default), **Date**, or **Time**. Select from preset date formats, including **Short** and **Long**. Choose from preset time formats, including **ShortTime** and **LongTime**. See [Edit Modes](#) (page 18), [Date Format](#) (page 18), and [Time Format](#) (page 18) for more information.

- **Supports Spin Buttons**

The **C1DateTimePicker** control supports spin (up/down) buttons for selecting time.

- **Supports a Wide Range of Cultures**

Define the cultural setting for date and time formats.

- **Supports Null Values**

The **C1DateTimePicker** control allows, by default, the entering of null values. This can be disabled by setting the **AllowNull** property to **False**.

DateTimePicker for WPF Quick Start

The following quick start guide is intended to get you up and running with **DateTimePicker for WPF**. In this quick start, you'll start in Visual Studio to create a new project, add a **C1DateTimePicker** control to your application, and customize the **C1DateTimePicker** control.

Step 1 of 3: Creating an Application with a C1DateTimePicker Control

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

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select **WPF Application**.
3. Enter a **Name** and **Location** for your project and click **OK** to create the new application.
4. In the Toolbox, double-click the **C1DateTimePicker** icon to add the **C1DateTimePicker** control to the WPF application.

You have completed the first step of the **DateTimePicker for WPF** quick start. In this step, you created a project and added a **C1DateTimePicker** control to it. In the next step, you'll customize the control.

Step 2 of 3: Customizing the Control

In the previous step, you created a WPF application with a **C1DateTimePicker** control. In this step, you will modify the appearance of the control.

Select the **C1DateTimePicker** control and then, in the Properties window, set the following properties:

- Set the **Height** property to "30" to set height of the control.

- Set the **Width** property to "300" to set the width of the control.
- Set the TimeFormat property to **ShortTime** to change the format of the time to a short format consisting of only hours and minute spaces.
- Set the DateFormat property to **Long** to change the format of the date to a longer format that includes the weekday.
- Set the SelectionBackground property to **LimeGreen** to modify the color of the control's selected area.
- Set the FirstDayOfWeek property to **Wednesday** to change the first day of the drop-down calendar's week to Wednesday.

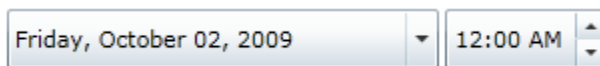
In this step, you customized the appearance of the C1DateTimePicker control. In the next step, you will run the project and experience the functionality of the control.

Step 3 of 3: Running the Project

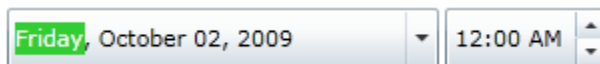
In the last step, you customized the C1DateTimePicker control. In this step, you will run the project and observe some of the run-time features of the control.

Complete the following steps:

1. From the toolbar menu, select **Project | Run Project** to run your application. Observe that the video content plays automatically and that the application resembles the following:



2. Using your cursor, highlight an area in the date picker. The selection resembles the following:



3. Click the time picker drop-down arrow to reveal the calendar and observe that the calendar's weeks start with Wednesday:



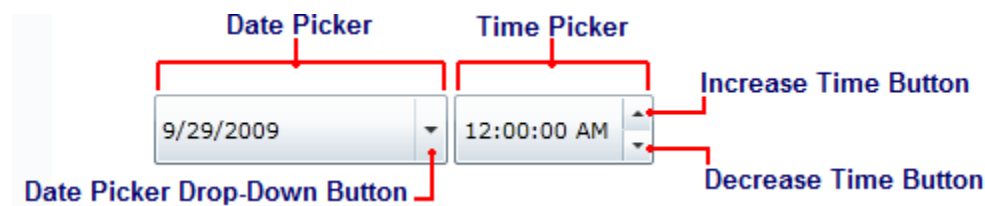
Congratulations! You have completed the **DateTimePicker for WPF** quick start. In this quick start, you created a WPF application containing a C1DateTimePicker control, modified the control's appearance, and experienced the control's run-time appearance features.

Working with C1DateTimePicker

The following topics introduce you to all of the elements and several features of the C1DateTimePicker control.

C1DateTimePicker Elements

ComponentOne DateTimePicker for WPF includes the C1DateTimePicker control, a simple control which provides, by default, both a date picker and a time picker. When you add the C1DateTimePicker control to a XAML window, it exists as a completely functional date and time picker. By default, the control's interface looks similar to the following image:



The C1DateTimePicker control consists of the following elements:

- **Date Picker**

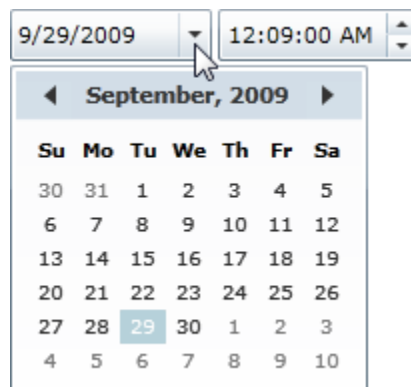
The date picker element is comprised of a date field and the calendar drop-down button. You can set the date by entering numeric values or by selecting a date from the calendar.

- **Time Picker**

The time picker element is comprised of the time field, the increase time button, and the decrease time button. You can set the time by entering numeric values or by clicking the buttons.

- **Date Picker Drop-Down Button**

The date picker drop-down button opens a calendar from where you can select a date for the date picker.



- **Increase Time Button**

The increase time button allows you to increase the time displayed in the time picker. Clicking the increase button will increase the time by one minute.

- **Decrease Time Button**

The decrease time button allows you to decrease the time displayed in the time picker. Clicking the decrease button will decrease the time by one minute.

Edit Modes

By default, the C1DateTimePicker control will appear on the page with both the date picker and the time picker. You can change the pickers that are displayed by setting the EditMode property to **Date**, **Time**, or **DateTime**. You can set the EditMode property to **Date** to display only the date picker; you can set the EditMode property to **Time** to only display the time picker; and you can set the EditMode property to **DateTime** to display both the time picker and date picker. The table below illustrates each editor mode.

Editor Mode	Result
Date	<input type="text" value="9/29/2009"/>
Time	<input type="text" value="12:09:00 AM"/>
DateTime	<input type="text" value="9/29/2009"/> <input type="text" value="12:09:00 AM"/>

Date Format

You can use the DateFormat property to set the format that the date picker displays. You can set DateFormat property to either **Short** or **Long**. The table below illustrates the two date formats.

Date Format	Result
Short (Default)	<input type="text" value="9/29/2009"/> <input type="text" value="12:00:00 AM"/>
Long	<input type="text" value="Tuesday, September 29, 2009"/> <input type="text" value="12:00:00 AM"/>

Time Format

You can use the TimeFormat property to set the format that the date picker displays. You can set TimeFormat property to either **ShortTime** or **LongTime**. The table below illustrates the two date formats.

Time Format	Result
ShortTime	<input type="text" value="9/29/2009"/> <input type="text" value="12:00 AM"/>
LongTime (default)	<input type="text" value="9/29/2009"/> <input type="text" value="12:00:00 AM"/>

DateTimePicker for WPF Layout and Appearance

The following topics detail how to customize the C1DateTimePicker control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

DateTimePicker for WPF Appearance Properties

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

Text Properties

The following properties let you customize the appearance of text in the C1DateTimePicker control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.

Color Properties

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

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

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

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.

BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.
---------------------------------	--

Size Properties

The following properties let you customize the size of the **C1DateTimePicker** control.

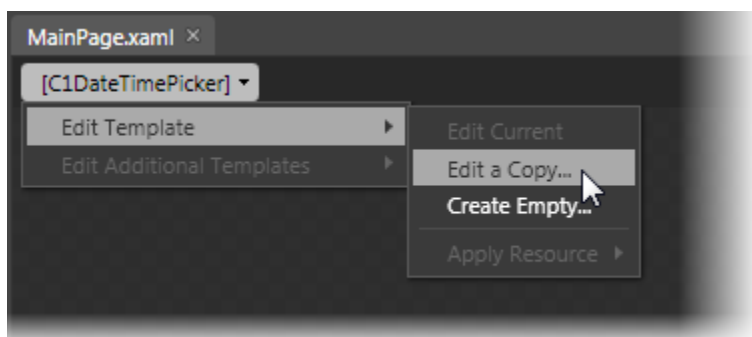
Property	Description
ActualHeight	Gets the rendered height of this element. This is a dependency property.
ActualWidth	Gets the rendered width of this element. This is a dependency property.
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets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

Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne DateTimePicker 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 C1DateTimePicker control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty** to create a new blank template.



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.

DateTimePicker 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 `C1DateTimePicker` control in general. If you are unfamiliar with the **ComponentOne DateTimePicker for WPF** product, please see the **DateTimePicker for WPF Quick Start** first.

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

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

Allowing Null Values

By default, the `C1DateTimePicker` control doesn't allow users to enter null values, but you can force the control to accept a null value by setting the `AllowNull` property to **True**. In this topic, you will learn how to set the `AllowNull` property to **True** in the designer, in XAML, and in code.

In the Designer

Complete the following steps:

1. Click the `C1DateTimePicker` control once to select it.
2. In the **Properties** window, select the `AllowNull` check box.

In XAML

To allow null values, place `AllowNull="True"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker AllowNull="True"/>
```

In Code

Complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Place the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1DateTimePicker1.AllowNull = True`
 - C#
`c1DateTimePicker1.AllowNull = true;`
3. Run the project.



Tip: to set null value at run-time, users should either clear all the text in the DatePicker portion of the control and press ENTER or move focus to the TimeEditor portion of the control.

Selecting the Edit Mode

By default, the `C1DateTimePicker` control shows both the date and time pickers, but you may also choose to show only the date picker or only the time picker. In this topic, you will learn how to change the editor mode in the designer, in XAML, and in code.

In the Designer

To change the edit mode, complete the following steps:

1. Click the `C1DateTimePicker` control once to select it.
2. In the **Properties** window, click the `EditMode` drop-down arrow and select a mode from the list. For this example, select **Date**.

In XAML

To change the edit mode, place `EditMode="Date"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker EditMode="Date">
```

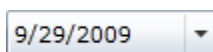
In Code

To change the edit mode, complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Import the following namespace:
 - Visual Basic
`Imports C1.WPF.DateTimeEditors`
 - C#
`using C1.WPF.DateTimeEditors;`
3. Place the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1DateTimePicker1.EditMode = C1DateTimePickerEditMode.Date`
 - C#
`c1DateTimePicker1.EditMode = C1DateTimePickerEditMode.Date;`
4. Run the project.

✔ This Topic Illustrates the Following:

In this topic, you set the `EditMode` to **Date**, which removes the time picker from the `C1DateTimePicker` control. The result of this topic will resemble the following image:



Selecting the Time Format

By default, the `C1DateTimePicker` control's time picker displays the time in a long format that includes seconds, but it can also display time in a shorter format. In this topic, you will learn how to change the time format in the designer, in XAML, and in code.

In the Designer

To change the time format, complete the following steps:

1. Click the `C1DateTimePicker` control once to select it.
2. In the **Properties** window, click the `TimeFormat` drop-down arrow and select a mode from the list. For this example, select **ShortTime**.

In XAML

To change the time format, place `TimeFormat="ShortTime"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker TimeFormat="ShortTime">
```

In Code

To change the time format, complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Import the following namespace:
 - Visual Basic
`Imports C1.WPF.DateTimeEditors`
 - C#
`using C1.WPF.DateTimeEditors;`
3. Place the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1DateTimePicker1.TimeFormat = C1TimeEditorFormat.ShortTime`
 - C#
`c1DateTimePicker1.TimeFormat = C1TimeEditorFormat.ShortTime;`
4. Run the project.



This Topic Illustrates the Following:

In this topic, you set the `TimeFormat` to **ShortTime**, which provides a shortened time display. The final result will resemble the following image:



Selecting the Date Format

By default, the `C1DateTimePicker` control's date picker displays the date in a short format, but it can also display time in a long format. In this topic, you will learn how to change the date format in the designer, in XAML, and in code.

In the Designer

To change the date format, complete the following steps:

1. Click the `C1DateTimePicker` control once to select it.
2. In the **Properties** window, click the `DateFormat` drop-down arrow and select a mode from the list. For this example, select **Long**.

In XAML

To change the date format, place `DateFormat="Long"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker DateFormat="Long">
```

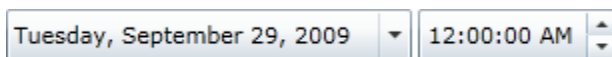
In Code

To change the date format, complete the following steps:

1. Open the **Window1.xaml.cs** page.
2. Place the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1DateTimePicker1.DateFormat =
Microsoft.Windows.Controls.DatePickerFormat.Long`
 - C#
`c1DateTimePicker1.DateFormat =
Microsoft.Windows.Controls.DatePickerFormat.Long;`
3. Run the project.

✓ This Topic Illustrates the Following:

In this topic, you set the **DateFormat** to **Long**, which will display the date in a long format. The final result will resemble the following image:



Setting the Minimum and Maximum Calendar Dates

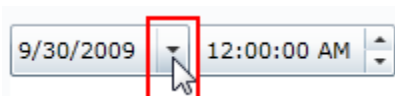
You can change the dates that calendar spans by setting the **MinimumDate** and **MaximumDate** properties in the designer, in XAML, and in code.

Note: Try to avoid setting the **MinDate** and **MaxDate** properties in XAML as a string value. Parsing these values from strings is culture specific. If you set a value with your current culture and a user is using different culture, the user can get **XamlParseException** when loading your site. The best practice is to set these values from code or via data binding.


In the Designer

Complete the following steps:

1. Click the **C1DateTimePicker** control once to select it.
2. In the Properties window, set the following properties:
 - Set the **MinDate** property to 01/01/2008.
 - Set the **MaxDate** property to 12/31/2012.
3. Run the program.
4. Click date picker drop-down button to expose the calendar.



5. Click the forward button to move the calendar ahead until you can no longer go forward. Note that the calendar stops on December 2012.

- Click the back button  to move the calendar back until you can no longer go forward. Note that the calendar stops on January 2008.

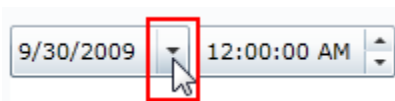
In XAML



Complete the following steps:

- Add `MinDate="2008-01-01"` and `MaxDate="2012-12-31"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker Height="26" Name="c1DateTimePicker1"
    MinDate="2008-01-01" MaxDate="2012-12-31" />
```

- Run the program.
- Click date picker drop-down button to expose the calendar.



- Click the forward button  to move the calendar ahead until you can no longer go forward. Note that the calendar stops on December 2012.
- Click the back button  to move the calendar back until you can no longer go forward. Note that the calendar stops on January 2008

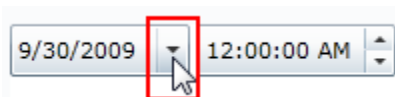
In Code

Complete the following:


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

```
'Set the minimum date
C1DateTimePicker1.MinDate = new DateTime(2008, 01, 01)
'Set the maximum date
C1DateTimePicker1.MaxDate = new DateTime(2012, 12, 31)
```
 - C#

```
//Set the minimum date
c1DateTimePicker1.MinDate = new DateTime(2008, 01, 01);
//Set the maximum date
c1DateTimePicker1.MaxDate = new DateTime(2012, 12, 31);
```
- Run the program.
- Click date picker drop-down button to expose the calendar.



- Click the forward button to move the calendar ahead until you can no longer go forward. Note that the calendar stops on December 2012.

- Click the back button  to move the calendar back until you can no longer go forward. Note that the calendar stops on January 2008.

Specifying the Date and Time

You can specify the time and date of a `C1DateTimePicker` control by setting the `DateTime` property using XAML or code.

Note: Try to avoid setting the `DateTime` property in XAML. Parsing these values from strings is culture specific. If you set a value with your current culture and a user is using different culture, the user can get `XamlParseException` when loading your site. The best practice is to set these values from code or via data binding.

In XAML

Complete the following steps:

- Place `DateTime="1/17/2010 11:04 AM"` to the `<my:C1DateTimePicker>` tag so that the markup resembles the following:

```
<my:C1DateTimePicker DateTime="1/17/2010 11:04 AM" />
```

- Run the project and observe that the `C1DateTimePicker` control shows the date and time as 01/17/2010 11:04:00 AM.

In Code

Complete the following steps:

- Open the **Window1.xaml.cs** page.
- Place the following code beneath the **InitializeComponent()** method:
 - Visual Basic
`C1DateTimePicker1.DateTime = New DateTime(2010, 1, 17, 11, 04, 0)`
 - C#
`c1DateTimePicker1.DateTime = new DateTime(2010, 1, 17, 11, 04, 0);`
- Run the project and observe that the `C1DateTimePicker` control shows the date and time as 01/17/2010 11:04:00 AM.

This Topic Illustrates the Following:

The result of this topic resembles the following image:

