ComponentOne

Windows for WPF

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third 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)

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

Table of Contents

ComponentOne Windows for WPF Overview

Replace standard browser dialog windows with **ComponentOne Window™ for WPF**. The **C1Window** control shows content in a floating window. Experience enhanced performance and ease-of-use with a rich object model and time-saving customization.

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:

Help with ComponentOne Studio for WPF

Getting Started

For information on installing **ComponentOne Studio for WPF**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for WPF.

What's New

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

Key Features

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

- **Modal and Modeless Dialog Windows**

  Decide whether users can interact with other windows while the dialog window is present through modal and modeless dialog windows.

- **Separate XAML Files to Define Window Objects**

  C1Window objects are not children of any elements on the page. Because of this, they are defined in separate XAML files whose root element is a **C1Window** object.

- **Resizable Windows**

  You can easily determine whether or not the window should be resized. Scroll bars are automatically added when the window becomes too small to show all of the content.

- **Window State**

  Windows for WPF can be minimized and restored to its original size; the developer can set the current state of the window. The developer can also configure the window so it cannot be maximized.

- **Window Elements**

You can edit the XAML file in Microsoft Expression Blend and add any elements you want. With **Windows for WPF**, creating and maintaining **C1Window** objects is very easy.

- **XBAP Support**

  XBAPs are WPF Applications built for the Web. **ComponentOne Windows for WPF** are supported in XBAP scenarios for both modal and modeless windows.

Windows for WPF Quick Start

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

Note that while this example uses Visual Studio, you should also similarly be able to complete this quick start in Microsoft Expression Blend.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a WPF application using **Windows for WPF**. In this step you'll add two buttons and a text box to the form. The buttons ill open modal and modal C1Window dialog windows.

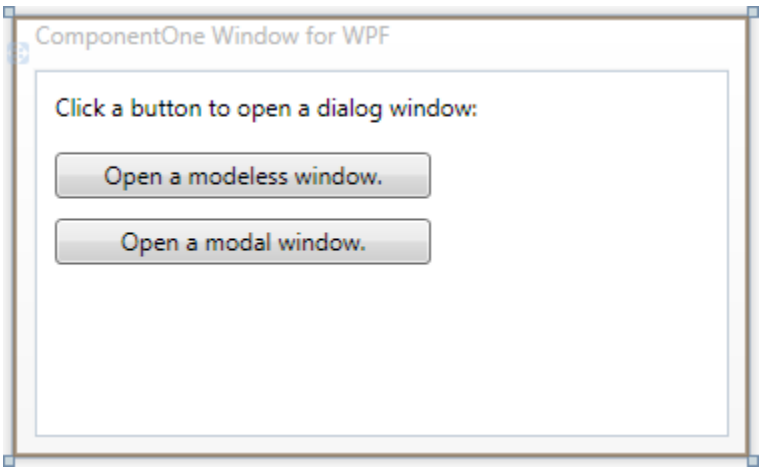
To set up your project and add a C1Window control to your application, complete the following steps:

1. In Visual Studio 2008, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **WPF Application**. Enter a **Name** for your project and click **OK**. The project will be created and a form will appear. For more information about creating a WPF project, see Creating a .NET Project in Visual Studio.
3. Right-click the project in the Solution Explorer window and select **Add Reference**.
4. In the **Add Reference** dialog box, locate and select the **C1.WPF.dll** assembly and click **OK** to add a reference to your project.
5. Click once on the form to select it, navigate to the Visual Studio Toolbox, and add a **TextBlock** and two **Button** controls (double-clicking on an item in the Toolbox to add it to the form).
6. Resize the form, and resize the controls on the form.
7. Select each control in turn, and set the following properties for each in the Properties window:
   - Set **TextBlock1**'s **Text** property to "Click a button to open a dialog window:"
   - Set **Button1**'s **Content** property to "Open a modeless window."
   - Set **Button2**'s **Content** property to "Open a modal window."

   The XAML markup will appear similar to the following:

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ComponentOne Windows for WPF" Height="220" Width="368">
    <Grid>
        <TextBlock Height="23" Margin="10,10,31,0" Name="TextBlock1"
VerticalAlignment="Top" Text="Click a button to open a dialog window:"
/>
        <Button Height="23" Margin="10,41,148,0" Name="Button1"
VerticalAlignment="Top">Open a modeless window.</Button>
        <Button Margin="10,74,148,0" Name="Button2" Height="23"
VerticalAlignment="Top">Open a modal window.</Button>
    </Grid>
</Window>
```

   The form will appear similar to the following image:



What You've Accomplished

You've successfully created a WPF application and added controls to the form. In the next step you'll add and customize the C1Window control.

Step 2 of 4: Adding C1Window Controls

In the previous step you created a new project and added button controls to the application. In this step you'll continue by adding a **C1Window** control in a user control.

Complete the following steps:

1. Right-click the project in the Visual Studio Solution Explorer and select the **Add | New Item** option. The **Add New Item** dialog box will open.
2. In the **Add New Item** dialog box, select the **WPF** item on the left and in the right Templates section choose **User Control (WPF)**, name the new control "MyWindow.xaml" and click **Add** to add the new user control. If it does not open automatically, double-click the **MyWindow.xaml** file in the Solution Explorer to open it.
3. Navigate to the Toolbox and double-click the **TextBlock** item to add the control to the form.
4. Set the **TextBlock**'s **Text** property to "Hello World!" The user control's XAML markup will now appear similar to the following:

```
<UserControl x:Class="MyWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="300"
Height="300">
    <Grid>
        <TextBlock Height="21" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="TextBlock1" VerticalAlignment="Top"
Width="120" Text="Hello World!" />
    </Grid>
</UserControl>
```

   Note that you can add additional controls to the form, if you choose – items you add will be included in the body of the **C1Window** control.

You've successfully set up your application's user interface, but if you run your application right now the buttons will do nothing when pressed. 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. Return to the Design view of the form.
2. Select **Button1** on the form, navigate to the Properties window, click the lightning bolt **Events** icon to view events, and click the space next to the **Click** item to create a **Button1_Click** event handler and switch to Code view.
3. Return to Design view and repeat the previous step with **Button2** to create the **Button2_Click** event handler.
4. In Code view, add the following import statement to the top of the page:
   - Visual Basic
     ```
     Imports C1.WPF
     ```
   - C#
     ```
     using C1.WPF;
     ```
5. Add the following code to the event handlers added earlier:
   - Visual Basic
     ```
     Private Sub Button1_Click(ByVal sender As Object, ByVal e As
     RoutedEventArgs)
         ShowWindow(False)
     End Sub

     Private Sub Button2_Click(ByVal sender As Object, ByVal e As
     RoutedEventArgs)
         ShowWindow(True)
     End Sub
     ```
   - C#
     ```
     void button1_Click(object sender, RoutedEventArgs e)
     {
       ShowWindow(false);
     }

     void button2_Click(object sender, RoutedEventArgs e)
     {
       ShowWindow(true);
     }
     ```
6. Add the following code below the **Button_Click** event handlers:
   - Visual Basic
     ```
     Private Sub ShowWindow(ByVal showModal As Boolean)
         Dim wnd As New C1Window()
         wnd.Header = "This is the header."
         wnd.Height = 120
         wnd.Width = 200
         wnd.Content = New MyWindow()
         wnd.CenterOnScreen()
         If showModal Then
             wnd.ShowModal()
         Else
     ```

```
        wnd.Show()
    End If
End Sub
```

- C#

```
private void ShowWindow(bool showModal)
{
    C1Window wnd = new C1Window();
    wnd.Header = "This is the header.";
    wnd.Height = 120;
    wnd.Width = 200;
    wnd.Content = new MyWindow();
    wnd.CenterOnScreen();
    if (showModal)
wnd.ShowModal();
    else
wnd.Show();
}
```

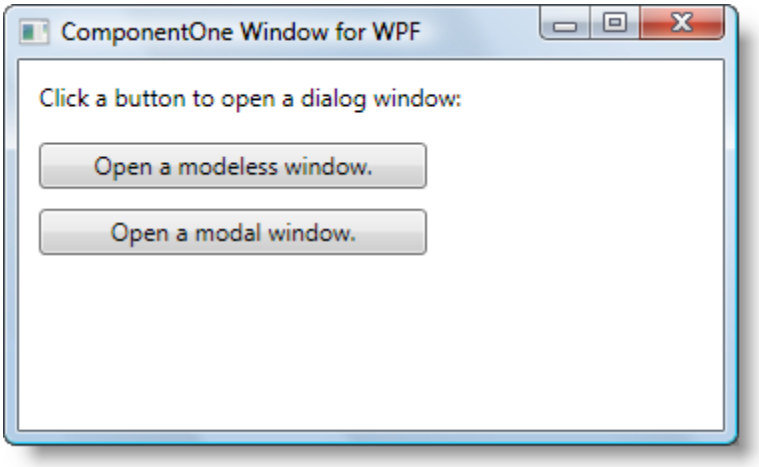
This code specifies the size of the window and opens a new window.

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

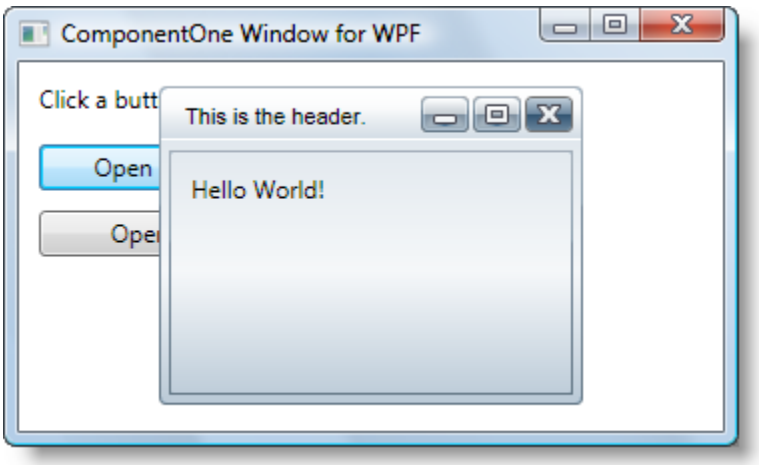
Step 4 of 4: Running the Application

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 **Windows 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:

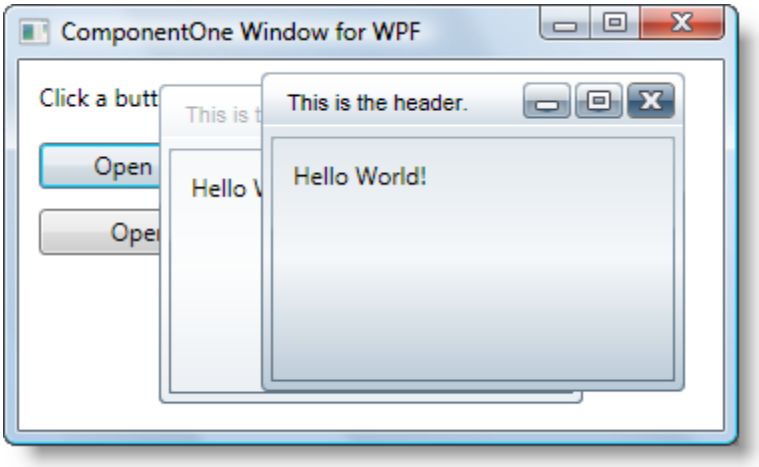


2. Click the **Open a modeless window** button. A modeless dialog window will open:

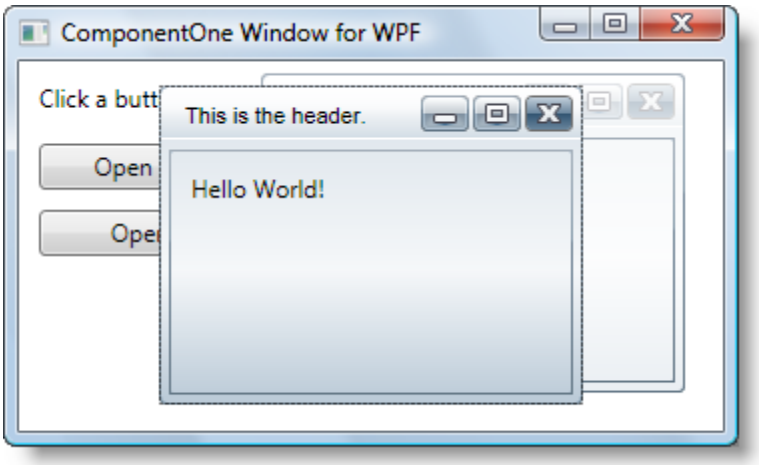


With modeless dialog windows, you can interact with other items on the page while the window is open.

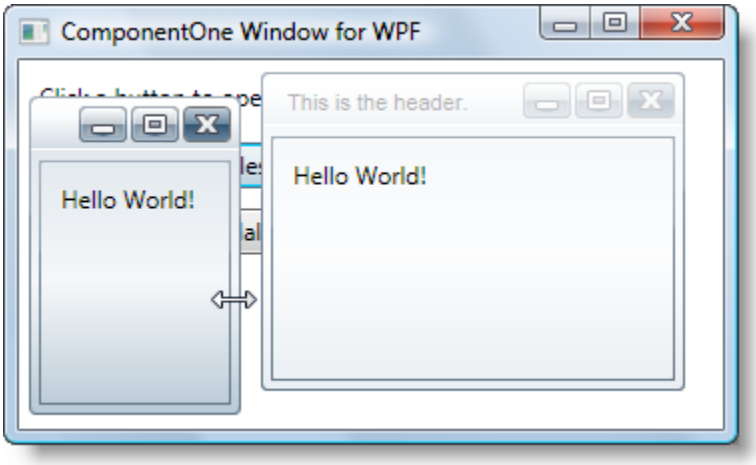
3. Click the **Open a modeless window** button again. Another modeless dialog window will open.



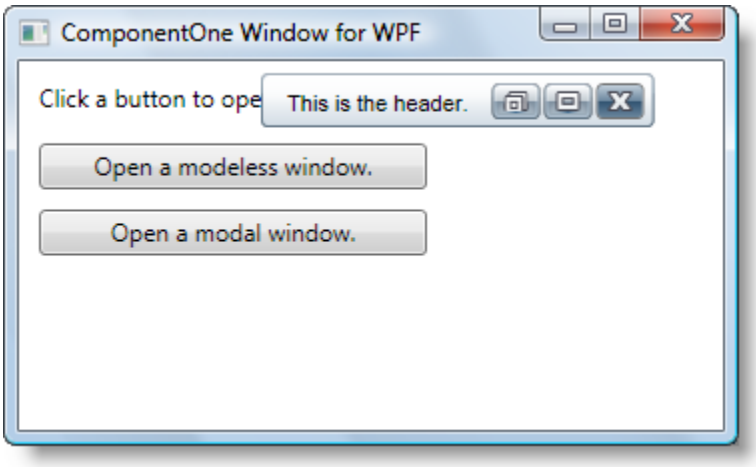
4. Click the first dialog box and notice that the focus shifts to it. It will appear like the following image:



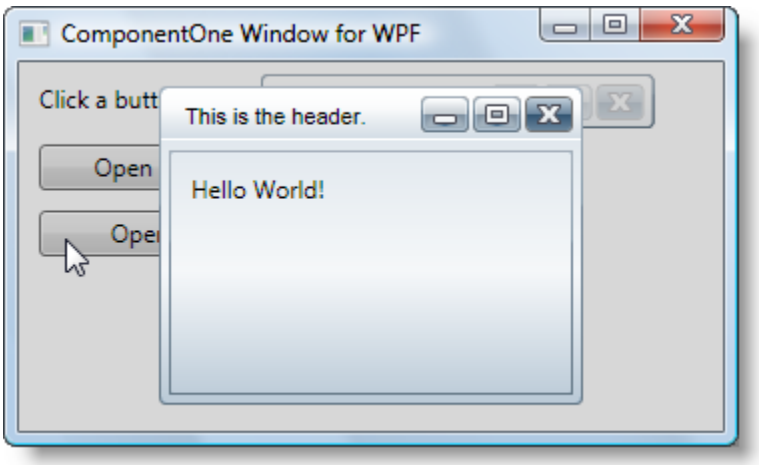
5. Click the dialog window's header and drag it to move it.
6. Click and drag a border to resize the dialog window. It will appear like the following image:



7. Click the "X" button to close the dialog window.
8. Click the "_" button to minimize the second dialog window. It will appear like the following image:



9. Click the **Open a modal window** button. A modal dialog window will open. It will appear like the following image:



Notice that except for the new dialog window the entire page is grayed out. With modal dialog windows, users will not be able to interact with any other element while the dialog window is open.

10. Try to select the minimized dialog window and notice that you cannot.
11. Click on the "X" close button to close the modal dialog box. Notice that you can now interact with elements on the page again.

Congratulations! You've completed the **Windows for WPF** quick start and created a simple WPF application, added controls, including a **C1Window** control, and viewed some of the run-time capabilities of **Windows for WPF**.

Modal and Modeless Dialog Windows

Dialog boxes are commonly used in applications to retrieve input from the user. In some applications a dialog box is used to prompt the user for input and once the application retrieves the input the dialog box is automatically closed or destroyed.

On the other hand, some applications use dialog boxes to display information while the user works in other windows. For example, when you check spelling in Microsoft Word a dialog box remains open so you can go through and edit your text in the document while the spell checker looks for the next misspelled word. To support the different ways applications use dialog boxes, **C1Window** supports two different types of dialog windows: modal (page 11) and modeless (page 11) dialog windows.

Modal Dialog Windows

A modal dialog window is a child window that must be closed before the user can continue working on the current application. Typically modal dialog windows either take control of the entire system or application displaying them until they are closed. For example, you can use a modal dialog window to retrieve login information from a user before the user can continue working on an application. Modal windows are useful in presenting important information or requiring user interaction.

You can show the C1Window control as a modal dialog box using the ShowModal method.

Modeless Dialog Windows

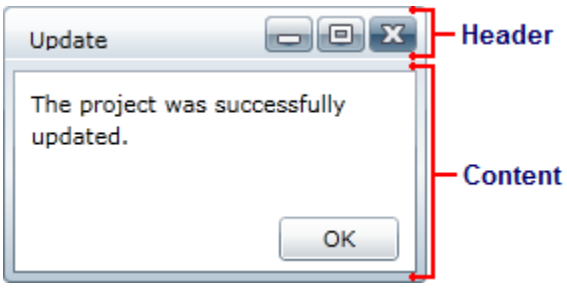
A modeless dialog window enables users to interact with other windows while the dialog window is present. Use this type of dialog window when the requested information is not necessary to continue. Modeless dialog windows do not keep the input focus so you can work on two applications at once.

A modeless dialog window is commonly used in menus and help systems where the user can use the dialog window and the application window concurrently. For example, a toolbar is a modeless dialog window because it can be detached from the application and the user can select items in the toolbar to apply features to the detached or separated application.

You can show the C1Window control as a modeless dialog box using the Show method.

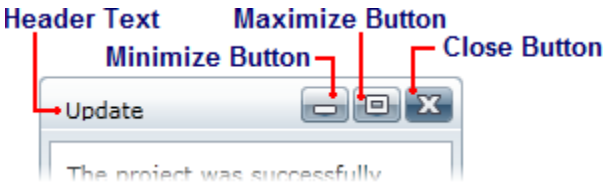
C1Window Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Window control. Like a typical dialog box, the **C1Window** control includes two main elements: a header and a content area:



Header

The header area includes the typical caption bar elements including title text and **Minimize**, **Maximize**, and **Close** buttons:



You can set the text in the header with the **Header** property. You can set the visibility of the buttons using the ShowCloseButton, ShowMaximizeButton, and ShowMinimizeButton properties. By default all three properties are **True** and the buttons are visible. You can also set the **Header** property to a **UIElement**. For example, this would allow you to add an icon next to the text in the caption bar.

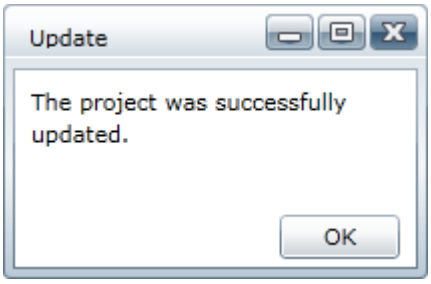
Content Area

The content area of the C1Window control can be set using the **Content** property. The content area can contain controls, a panel (such as a **Grid** or **StackPanel**) that contains controls, text, or other elements. You can, for example, set the **Content** property to a **UserControl** that contains many controls and elements.

Working with Windows for WPF

ComponentOne Windows for WPF includes the C1Window control, a simple dialog window control that shows content in a floating window. When added to your application, the C1Window control appears similar to a traditional dialog box within a WPF application.

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



Basic Properties

ComponentOne Windows 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 Window Appearance Properties (page 14) for more information about properties that control appearance.

The following properties let you customize the C1Window control:

Property	Description
Content	Gets or sets the content of a ContentControl. This is a dependency property.
ContentTemplate	Gets or sets the data template used to display the content of the ContentControl. This is a dependency property.

DialogResult	Gets or sets the dialog result for the window.
Header	Gets or sets the header of this control.
HeaderTemplate	Gets or sets the data template used to display the header.
IsResizable	Gets or sets whether the window can be resized and maximized.
Language	Gets or sets localization/globalization language information that applies to an element.
ModalBackground	Gets or sets the brushed used on the background when showing a modal window.
ShowCloseButton	Gets or sets whether the close button of this window is shown.
ShowMaximizeButton	Gets or sets whether the maximize button of this window is shown.
ShowMinimizeButton	Gets or sets whether the minimize button of this window is shown.
WindowState	Gets or sets a value that indicates whether a window is restored, minimized, or maximized.

Basic Events

ComponentOne Windows 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 C1Window control:

Event	Description
Closed	Event fired when the window is closed by the user or the Close() method.
Closing	Event fired when the window is about to close, allows the handler to stop the window from being closed.
PositionChanged	Fires when the window position changes.
WindowStateChanged	Event raised when the WindowState property has changed

Basic Methods

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

The following methods let you customize the C1Window control:

Event	Description
BringToFront	Puts the window in front of all windows.
CenterOnScreen	Centers the window in its container.
Close	Closes the window.
Hide	Hides the window without closing it.
Show	Opens the window as a modeless dialog window.

ShowModal	Opens the window as a modal dialog window.

Window State

The C1Window dialog window can be floating, maximized, or minimized. You can customize the dialog window's state by setting the WindowState property to one of the following options:

Event	Description
Floating	The window is floating (neither maximized nor minimized).
Maximized	The window is maximized.
Minimized	The window is minimized.

Window Layout and Appearance

The following topics detail how to customize the C1Window 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.

Window Appearance Properties

ComponentOne Windows 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
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.
ModalBackground	Gets or sets the brushed used on the background when showing a modal window. For an example, see Setting the Modal Background Color (page 25).

Alignment Properties

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

Property	Description
HorizontalAlignment	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.
VerticalAlignment	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
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 **C1Window** control:

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

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 C1Window control as well as a description of the property:

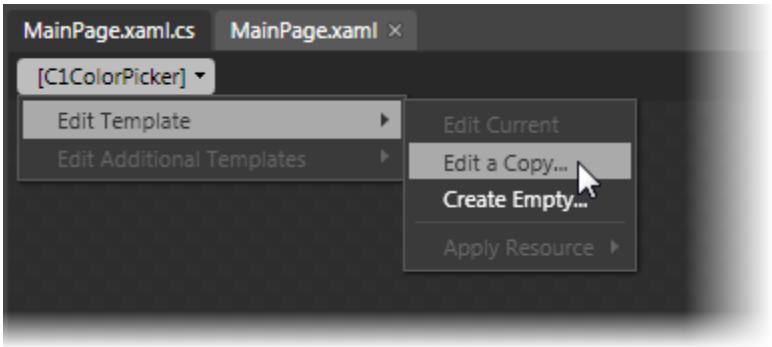
Property	Description
Background	Gets or sets a brush that describes the background of a control. The default **Background** color is LightBlue.
ButtonBackground	A brush used to define the appearance of the background of the control.
ButtonForeground	A brush used to define the appearance of the foreground of the control.
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.

Window 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 Windows 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 C1Window 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.

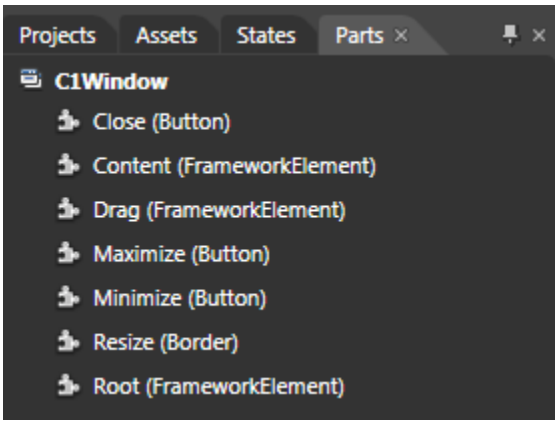
Window Styles

ComponentOne Windows for WPF's **C1Window** 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
FontStyle	Gets or sets the font style. This is a dependency property.
HeaderFontStyle	Gets or sets the font style of the header.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Window Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1Window** 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** pane will change to indicate selection.

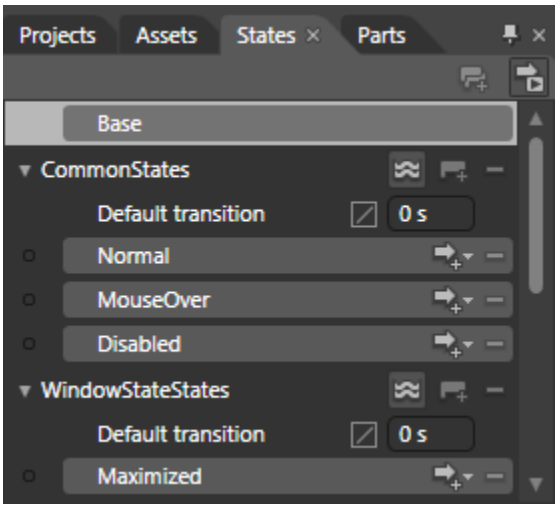
Template parts available in the **C1Window** control include:

Name	Type	Description
Close	Button	Represents a button control used to close the dialog window.
Content	FrameworkElement	This element contains the window's content.
Drag	FrameworkElement	This element listens for mouse events to drag the window.

Maximize	Button	Represents a button control used to maximize the dialog window.
Minimize	Button	Represents a button control used to minimize the dialog window.
Resize	Border	Draws a border, background, or both around another object. Here the border is used to resize the control.
Root	FrameworkElement	The root element of the template.

Window 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 17). 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. Window state states include **Maximized** for when the window is maximized, **Minimized** for when the window is minimized, and **Floating** for when the window is neither maximized nor minimized.

Active states include **Active** when the window is active and in focus and **Inactive** for when the window is inactive and not in focus. Drag states include **Still** for when the window is not being dragged, and **Dragged** when the user is in the process of completing a drag-and-drop operation with the window.

XAML Elements

Several auxiliary XAML elements are installed with **ComponentOne Windows for WPF**. These elements include templates and themes and are located in the **Windows 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 **Windows for WPF**:

Element	Folder	Description

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

Windows 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 **Windows** page in the **ControlExplorer** sample demonstrates showing a modal or modeless window and how to customize the C1Window control.

Windows 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 C1Window control in general. If you are unfamiliar with the **ComponentOne Windows for WPF** product, please see the Windows for WPF Quick Start (page 3) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Windows 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 or Creating a Microsoft Blend Project.

Setting the Title Text

You can easily customize **Windows for WPF** control's header area by adding a title. For more information the header area, see C1Window Elements (page 11). By default the window does not include text in the caption bar, but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code by setting the **Header** property.

At Design Time in Blend

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

1. Click the C1Window control once to select it.
2. Navigate to the Properties tab and locate the **Header** item.
3. Click in the text box next to the **Header** item, and enter "Hello World!" or some other text.

   This will set the **Header** property and the text in the caption bar of the dialog window to the text you chose.

In XAML

For example, to set the **Header** property add `Header="Hello World!"` to the `<c1:C1Window>` tag so that it appears similar to the following:

```
<c1:C1Window Height="110" HorizontalAlignment="Right" Margin="0,54,71,0"
VerticalAlignment="Top" Width="220" Content="C1Window" Header="Hello
World!"/>
```

In Code

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

- Visual Basic

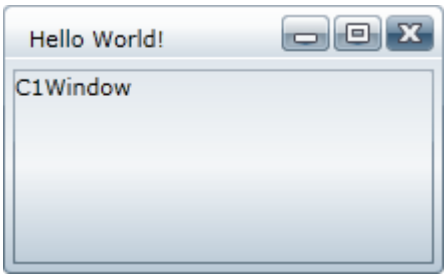
```
Me.C1Window1.Header = "Hello World!"
```

- C#

```
this.c1window1.Header = "Hello World!";
```

Run the application and observe:

The caption bar of the C1Window control will appear with "Hello World!" or the text you chose:



Hiding Header Buttons

You can easily customize user interaction by setting what buttons are visible on the **Windows for WPF** control's caption bar. For more information about the header area, see C1Window Elements (page 11). By default the window displays **Minimize**, **Maximize**, and **Close** buttons, but you can customize this in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To hide the **Maximize** and **Minimize** buttons in Blend, complete the following steps:

1. Click the C1Window control once to select it.
2. Navigate to the Properties window tab.
3. Locate the **ShowMaximizeButton** item and clear the check box next to the item.
4. Locate the **ShowMinimizeButton** item and clear the check box next to the item.

   This will hide the **Maximize** and **Minimize** buttons.

In XAML

For example, to remove the **Maximize** and **Minimize** add `ShowMaximizeButton="False" ShowMinimizeButton="False"` to the `<c1:C1Window>` tag so that it appears similar to the following:

```
<c1:C1Window Height="129" HorizontalAlignment="Right" Margin="0,54,71,0"
VerticalAlignment="Top" Width="220" Content="C1Window"
ShowMaximizeButton="False" ShowMinimizeButton="False"/>
```

In Code

For example, to hide the **Maximize** and **Minimize** buttons, add the following code to your project:

- Visual Basic

```
Me.C1Window1.ShowMaximizeButton = False
Me.C1Window1.ShowMinimizeButton = False
```

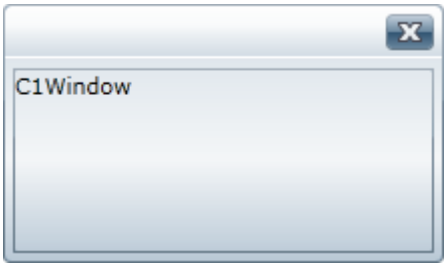
- C#

```
this.c1window1.ShowMaximizeButton = false;
```

```
this.c1window1.ShowMinimizeButton = false;
```

Run the application and observe:

The caption bar of the C1Window control will appear without the **Maximize** and **Minimize** buttons displayed:



Note that to hide the **Close** button, you can set the ShowCloseButton property.

Minimizing and Maximizing the Window

The C1Window control includes caption bar buttons that will minimize or maximize the control. However you can also customize the appearance of the control using the WindowState property. In this topic you'll add a C1Window control to the form and three buttons: one that will minimize the window, one that will maximize the window, and one that will restore a minimized or maximized window.

Complete the following steps:

1. Create a new WPF project in Microsoft Expression Blend.
2. Click the **Assets** button in the Toolbar (the double-chevron icon).
3. In the dialog box locate and select the **C1Window** icon.

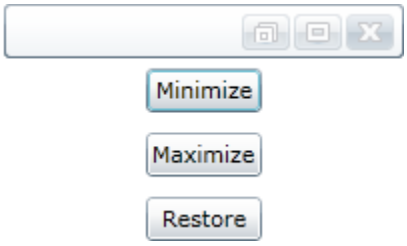
   Note that if you cannot locate the **C1Window** icon in the dialog box, you may need to add a reference to the **C1.WPF** assembly first.
4. Select the page and double-click the **C1Window** icon in the Toolbar. The dialog window will be added to the page.
5. Select the **C1Window** control, navigate to the Properties window, and set the following properties:
   - Set the control's **Name** property to "window".
   - Set the control's **Height** to **150** and **Width** to **200**.
   - Set the control's **Margin** property to **5**.
   - Set the control's **Content** property to " Minimize or Maximize me!".
6. In the Toolbox, click the Panels icon (by default a Grid) and select the **StackPanel** item.
7. Double-click the **StackPanel** to add one to the application.
8. Select the **StackPanel** and in the Properties window set its **HorizontalAlignment** and **VerticalAlignment** properties to **Center**.
9. Set the **StackPanel**'s **Height** and **Width** properties to **Auto**.
10. Double-click the **Button** icon in the Toolbox three times to add three **Button** controls to the **StackPanel** below the **C1Window** control.
11. Select each of the buttons in turn and set the following properties in the Properties window:

- Set the first button's **Name** and **Content** properties to "Minimize".
- Set the second button's **Name** and **Content** properties to "Maximize".
- Set the third button's **Name** and **Content** properties to "Restore".

12. Set the **Margin** property for each of the buttons to **5**.
13. Select the **Minimize** button, click the lightning bolt **Events** icon in the Properties window.
14. Double-click the space next to the **Click** event to create an event handler and switch to Code view. Return to Design view and repeat this step with the remaining buttons to create a **Button_Click** event handler for each one.
15. In Code view, add the following import statement to the top of the page:
    - Visual Basic
      ```
      Imports C1.WPF
      ```
    - C#
      ```
      using C1.WPF;
      ```
16. Add code to the **Button_Click** event handlers you created earlier. They will appear similar to the following:
    - Visual Basic
      ```
      Private Sub Minimize_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Minimized
      End Sub

      Private Sub Maximize_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Maximized
      End Sub

      Private Sub Restore_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Floating
      End Sub
      ```
    - C#
      ```
      private void Minimize_Click(object sender,
      System.Windows.RoutedEventArgs e)
      {
          this.window.WindowState = C1WindowState.Minimized;
      }

      private void Maximize_Click(object sender,
      System.Windows.RoutedEventArgs e)
      {
          this.window.WindowState = C1WindowState.Maximized;
      }

      private void Restore_Click(object sender,
      System.Windows.RoutedEventArgs e)
      {
          this.window.WindowState = C1WindowState.Floating;
      }
      ```

Run the application and observe:

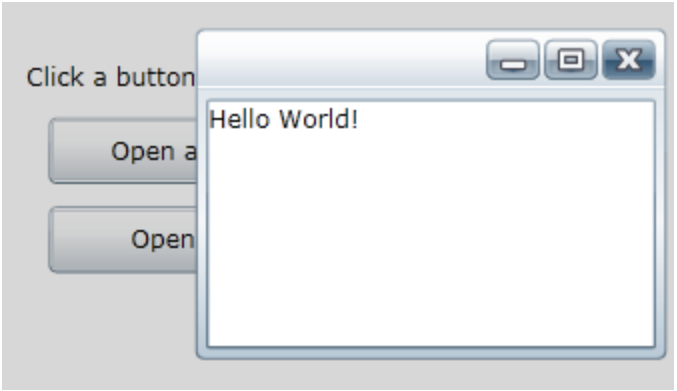
A dialog box appears with three buttons on the page. You can click the **Minimize** button to minimize the **C1Window** control:



You can maximize the **C1Window** dialog window control by clicking the **Maximize** button, and you can return the **C1Window** control to its original appearance by clicking the **Restore** button.

Setting the Modal Background Color

At run time when you open a modal dialog window, you will notice that the area behind the window is grayed out. This indicates that the dialog window must be closed before the user can interact with elements on the page. For example:



You can customize this background color by setting the ModalBackground property. For an example, complete the following steps:

1. Create an application that includes a button that opens a modal dialog window named "window". For example, complete the steps in the outlined in the Windows for WPF Quick Start (page 3).
2. In Code view, add the following code to the **Button_Click** event:
   - Visual Basic
     ```
     Dim bgcol As New SolidColorBrush()
     bgcol.Color = Color.FromArgb(150, 255, 0, 0)
     window.ModalBackground = bgcol
     ```
   - C#
     ```
     SolidColorBrush bgcol = new SolidColorBrush();
     bgcol.Color = Color.FromArgb(150, 255, 0, 0);
     window.ModalBackground = bgcol;
     ```

This code will create a new red-colored brush and set the ModalBackground property to the brush. The code in the **Button_Click** event will now appear similar to the following:

- Visual Basic

```
Private Sub ShowDialog(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    Dim window = New C1Window()
    window.Content = New MyWindow()
    window.CenterOnScreen()
    Dim bgcol As New SolidColorBrush()
    bgcol.Color = Color.FromArgb(150, 255, 0, 0)
    window.ModalBackground = bgcol
    window.ShowModal()
End Sub
```

- C#

```
void ShowDialog(object sender, RoutedEventArgs e)
{
    var window = new C1Window();
    window.Content = new MyWindow();
    window.CenterOnScreen();
    SolidColorBrush bgcol = new SolidColorBrush();
    bgcol.Color = Color.FromArgb(150, 255, 0, 0);
    window.ModalBackground = bgcol;
    window.ShowModal();
}
```

Run the application and observe:

Run the application and open the dialog window as a modal dialog box. You will notice that the background color behind the window appears red or the color you chose: