**Introduction to Windows Presentation Foundation**

Windows Presentation Foundation (WPF) is a next-generation presentation system for building Windows client applications with visually stunning user experiences. With WPF, you can create a wide range of both standalone and browser-hosted applications.

The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography. WPF is included in the Microsoft .NET Framework, so you can build applications that incorporate other elements of the .NET Framework class library.

This overview is intended for newcomers and covers the key capabilities and concepts of WPF. Experienced WPF developers seeking a review of WPF may also find this overview useful.

|  |
| --- |
| **Note:** |
| For new and updated WPF features in the .NET Framework 3.5, see [What's New in Windows Presentation Foundation Version 3.5](http://msdn.microsoft.com/en-us/library/bb613588.aspx). |

This topic contains the following sections.

* [Programming with WPF](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Programming_with_WPF#Programming_with_WPF)
* [Markup and Code-Behind](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Markup_And_Codebehind#Markup_And_Codebehind)
* [Applications](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Applications#Applications)
* [Controls](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Controls#Controls)
* [Input and Commanding](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Input_And_Commanding#Input_And_Commanding)
* [Layout](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Layout#Layout)
* [Data Binding](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Data_Binding#Data_Binding)
* [Graphics](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Graphics#Graphics)
* [Animation](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Animation#Animation)
* [Media](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Media#Media)
* [Text and Typography](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Text_and_Typography#Text_and_Typography)
* [Documents](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Documents#Documents)
* [Customizing WPF Applications](http://msdn.microsoft.com/en-us/library/aa970268.aspx#WPF_Customization#WPF_Customization)
* [WPF Best Practices](http://msdn.microsoft.com/en-us/library/aa970268.aspx#WPF_Best_Practices#WPF_Best_Practices)
* [Summary](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Summary#Summary)
* [Recommended Overviews and Samples](http://msdn.microsoft.com/en-us/library/aa970268.aspx#Recommended_WPF_Topic_Shortcuts#Recommended_WPF_Topic_Shortcuts)
* [Related Topics](http://msdn.microsoft.com/en-us/library/aa970268.aspx#seeAlsoSection#seeAlsoSection)

 Programming with WPF

WPF exists as a subset of .NET Framework types that are for the most part located in the [System.Windows](http://msdn.microsoft.com/en-us/library/system.windows.aspx) namespace. If you have previously built applications with .NET Framework using managed technologies like ASP.NET and Windows Forms, the fundamental WPF programming experience should be familiar; you instantiate classes, set properties, call methods, and handle events, all using your favorite .NET Framework programming language, such as C# or Visual Basic.

To support some of the more powerful WPF capabilities and to simplify the programming experience, WPF includes additional programming constructs that enhance properties and events: dependency properties and routed events. For more information on dependency properties, see [Dependency Properties Overview](http://msdn.microsoft.com/en-us/library/ms752914.aspx). For more information on routed events, see [Routed Events Overview](http://msdn.microsoft.com/en-us/library/ms742806.aspx).

 Markup and Code-Behind

WPF offers additional programming enhancements for Windows client application development. One obvious enhancement is the ability to develop an application using both markup and code-behind, an experience that ASP.NET developers should be familiar with. You generally use Extensible Application Markup Language (XAML) markup to implement the appearance of an application while using managed programming languages (code-behind) to implement its behavior. This separation of appearance and behavior has the following benefits:

* Development and maintenance costs are reduced because appearance-specific markup is not tightly coupled with behavior-specific code.
* Development is more efficient because designers can implement an application's appearance simultaneously with developers who are implementing the application's behavior.
* Multiple design tools can be used to implement and share XAML markup, to target the requirements of the application development contributors; [Microsoft Expression Blend](http://go.microsoft.com/fwlink?LinkID=88924) provides an experience that suits designers, while Visual Studio 2005 targets developers.
* Globalization and localization for WPF applications is greatly simplified (see [WPF Globalization and Localization Overview](http://msdn.microsoft.com/en-us/library/ms788718.aspx)).

The following is a brief introduction to WPF markup and code-behind. For more information on this programming model, see [XAML Overview](http://msdn.microsoft.com/en-us/library/ms752059.aspx) and [Code-Behind and XAML](http://msdn.microsoft.com/en-us/library/aa970568.aspx).

### Markup

XAML is an XML-based markup language that is used to implement an application's appearance declaratively. It is typically used to create windows, dialog boxes, pages, and user controls, and to fill them with controls, shapes, and graphics.

The following example uses XAML to implement the appearance of a window that contains a single button.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

Title="Window with Button"

Width="250" Height="100">

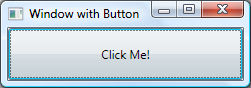
<!-- Add button to window -->

<Button Name="button">Click Me!</Button>

</Window>

Specifically, this XAML defines a window and a button by using the Window and Button elements, respectively. Each element is configured with attributes, such as the Window element's Title attribute to specify the window's title-bar text. At run time, WPF converts the elements and attributes that are defined in markup to instances of WPF classes. For example, the Window element is converted to an instance of the [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx) class whose [Title](http://msdn.microsoft.com/en-us/library/system.windows.window.title.aspx) property is the value of the Title attribute.

The following figure shows the user interface (UI) that is defined by the XAML in the previous example.



For more information, see [XAML Overview](http://msdn.microsoft.com/en-us/library/ms752059.aspx).

Since XAML is XML-based, the UI that you compose with it is assembled in a hierarchy of nested elements known as an element tree. The element tree provides a logical and intuitive way to create and manage UIs. For more information, see [Trees in WPF](http://msdn.microsoft.com/en-us/library/ms753391.aspx).

### Code-Behind

The main behavior of an application is to implement the functionality that responds to user interactions, including handling events (for example, clicking a menu, tool bar, or button) and calling business logic and data access logic in response. In WPF, this behavior is generally implemented in code that is associated with markup. This type of code is known as code-behind. The following example shows the code-behind and updated markup from the previous example.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.AWindow"

Title="Window with Button"

Width="250" Height="100">

<!-- Add button to window -->

<Button Name="button" Click="button\_Click">Click Me!</Button>

</Window>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl63VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl63VisualBasic');)

Namespace SDKSample

Partial Public Class AWindow

Inherits System.Windows.Window

Public Sub New()

' InitializeComponent call is required to merge the UI

' that is defined in markup with this class, including

' setting properties and registering event handlers

InitializeComponent()

End Sub

Private Sub button\_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)

' Show message box when button is clicked

MessageBox.Show("Hello, Windows Presentation Foundation!")

End Sub

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl64CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl64CSharp');)

using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample

{

public partial class AWindow : Window

{

public AWindow()

{

// InitializeComponent call is required to merge the UI

// that is defined in markup with this class, including

// setting properties and registering event handlers

InitializeComponent();

}

void button\_Click(object sender, RoutedEventArgs e)

{

// Show message box when button is clicked

MessageBox.Show("Hello, Windows Presentation Foundation!");

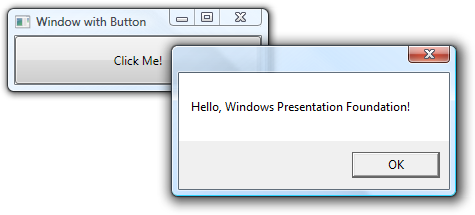
}

}

}

In this example, the code-behind implements a class that derives from the [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx) class. The x:Class attribute is used to associate the markup with the code-behind class. InitializeComponent is called from the code-behind class's constructor to merge the UI that is defined in markup with the code-behind class. (InitializeComponent is generated for you when your application is built, which is why you don't need to implement it manually.) The combination of x:Class and InitializeComponent ensure that your implementation is correctly initialized whenever it is created. The code-behind class also implements an event handler for the button's [Click](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.buttonbase.click.aspx) event. When the button is clicked, the event handler shows a message box by calling the [MessageBox..::.Show](http://msdn.microsoft.com/en-us/library/system.windows.messagebox.show.aspx) method.

The following figure shows the result when the button is clicked.



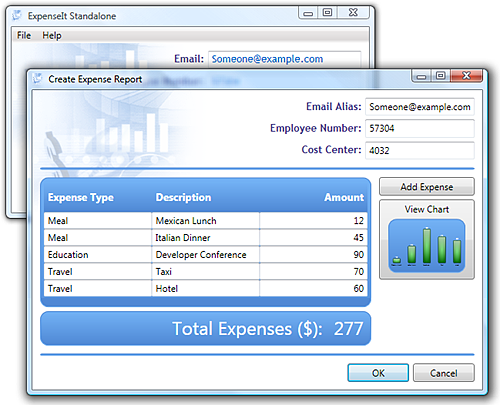
For more information, see [Code-Behind and XAML](http://msdn.microsoft.com/en-us/library/aa970568.aspx).

 Applications

.NET Framework, [System.Windows](http://msdn.microsoft.com/en-us/library/system.windows.aspx), and markup and code-behind, constitute the foundation of the WPF application development experience. Additionally, WPF has comprehensive features for creating user experiences with rich content. To package this content and deliver it to users as "applications," WPF provides types and services that are collectively known as the application model. The application model supports the development of both standalone and browser-hosted applications.

### Standalone Applications

For standalone applications, you can use the [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx) class to create windows and dialog boxes that are accessed from menu bars and tool bars. The following figure shows a standalone application with a main window and a dialog box.

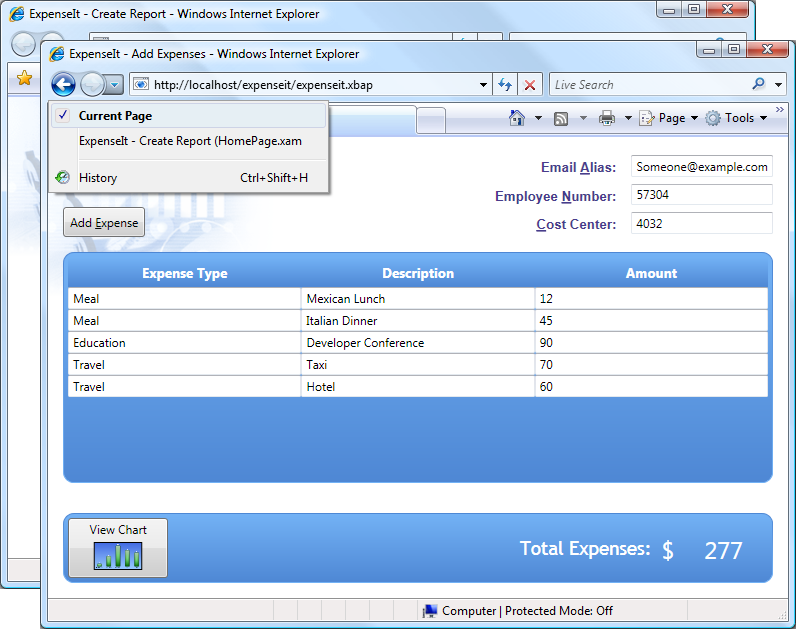


Additionally, you can use the following WPF dialog boxes: [MessageBox](http://msdn.microsoft.com/en-us/library/system.windows.messagebox.aspx), [OpenFileDialog](http://msdn.microsoft.com/en-us/library/microsoft.win32.openfiledialog.aspx), [SaveFileDialog](http://msdn.microsoft.com/en-us/library/microsoft.win32.savefiledialog.aspx), and [PrintDialog](http://msdn.microsoft.com/en-us/library/system.windows.controls.printdialog.aspx).

For more information, see [WPF Windows Overview](http://msdn.microsoft.com/en-us/library/ms748948.aspx).

### Browser-Hosted Applications

For browser-hosted applications, known as XAML browser applications (XBAPs), you can create pages ([Page](http://msdn.microsoft.com/en-us/library/system.windows.controls.page.aspx)) and page functions ([PageFunction<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/ms615540.aspx)) that you can navigate between using hyperlinks ([Hyperlink](http://msdn.microsoft.com/en-us/library/system.windows.documents.hyperlink.aspx) classes). The following figure shows a page in an XBAP that is hosted in Internet Explorer 7.



WPF applications can be hosted in both Microsoft Internet Explorer 6 and Internet Explorer 7. WPF offers the two following options for alternative navigation hosts:

* [Frame](http://msdn.microsoft.com/en-us/library/system.windows.controls.frame.aspx), to host islands of navigable content in either pages or windows.
* [NavigationWindow](http://msdn.microsoft.com/en-us/library/system.windows.navigation.navigationwindow.aspx), to host navigable content in an entire window.

For more information, see [Navigation Overview](http://msdn.microsoft.com/en-us/library/ms750478.aspx).

### The Application Class

Both XBAPs and standalone applications are often complex enough to require additional application-scoped services, including startup and lifetime management, shared properties, and shared resources. The [Application](http://msdn.microsoft.com/en-us/library/system.windows.application.aspx) class encapsulates these services and more, and it can be implemented by just using XAML, as shown in the following example.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl117CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl117CSharp');)

<Application

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

StartupUri="MainWindow.xaml" />

This markup is the application definition for a standalone application, and instructs WPF to create an [Application](http://msdn.microsoft.com/en-us/library/system.windows.application.aspx) object that automatically opens [MainWindow](http://msdn.microsoft.com/en-us/library/system.windows.application.mainwindow.aspx) when the application is started.

A key concept to understand about [Application](http://msdn.microsoft.com/en-us/library/system.windows.application.aspx) is that it provides a common platform of support for both standalone and browser-hosted applications. For example, the preceding XAML could be used by a browser-hosted application to automatically navigate to a page when an XBAP is started, as shown in the following example.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl121CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl121CSharp');)

<Application

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

StartupUri="HomePage.xaml" />

For more information, see [Application Management Overview](http://msdn.microsoft.com/en-us/library/ms743714.aspx).

### Security

Because XBAPs are hosted in a browser, security is important. In particular, a partial-trust security sandbox is used by XBAPs to enforce restrictions that are less than or equal to the restrictions imposed on HTML-based applications. Furthermore, each HTML feature that is safe to run from XBAPs in partial trust has been tested using a comprehensive security process, detailed in [Windows Presentation Foundation Security Strategy - Security Engineering](http://msdn.microsoft.com/en-us/library/aa969774.aspx).

Still, a majority of WPF features can be safely executed from XBAPs, as described in [Windows Presentation Foundation Partial Trust Security](http://msdn.microsoft.com/en-us/library/aa970910.aspx).

 Controls

The user experiences that are delivered by the application model are constructed controls. In WPF, "control" is an umbrella term that applies to a category of WPF classes that are hosted in either a window or a page, have a user interface (UI), and implement some behavior.

For more information, see [Controls Overview](http://msdn.microsoft.com/en-us/library/bb613551.aspx). For introductory samples, see [Control Library Samples](http://msdn.microsoft.com/en-us/library/ms771720.aspx)

### WPF Control Roll Call

The built-in WPF controls are listed here.

* Buttons: [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) and [RepeatButton](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.repeatbutton.aspx).
* Dialog Boxes: [OpenFileDialog](http://msdn.microsoft.com/en-us/library/microsoft.win32.openfiledialog.aspx), [PrintDialog](http://msdn.microsoft.com/en-us/library/system.windows.controls.printdialog.aspx), and [SaveFileDialog](http://msdn.microsoft.com/en-us/library/microsoft.win32.savefiledialog.aspx).
* Digital Ink: [InkCanvas](http://msdn.microsoft.com/en-us/library/system.windows.controls.inkcanvas.aspx) and [InkPresenter](http://msdn.microsoft.com/en-us/library/system.windows.controls.inkpresenter.aspx).
* Documents: [DocumentViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx), [FlowDocumentPageViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.flowdocumentpageviewer.aspx), [FlowDocumentReader](http://msdn.microsoft.com/en-us/library/system.windows.controls.flowdocumentreader.aspx), [FlowDocumentScrollViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.flowdocumentscrollviewer.aspx), and [StickyNoteControl](http://msdn.microsoft.com/en-us/library/system.windows.controls.stickynotecontrol.aspx).
* Input: [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx), [RichTextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.richtextbox.aspx), and [PasswordBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.passwordbox.aspx).
* Layout: [Border](http://msdn.microsoft.com/en-us/library/system.windows.controls.border.aspx), [BulletDecorator](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.bulletdecorator.aspx), [Canvas](http://msdn.microsoft.com/en-us/library/system.windows.controls.canvas.aspx), [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx), [Expander](http://msdn.microsoft.com/en-us/library/system.windows.controls.expander.aspx), [Grid](http://msdn.microsoft.com/en-us/library/system.windows.controls.grid.aspx), [GridView](http://msdn.microsoft.com/en-us/library/system.windows.controls.gridview.aspx), [GridSplitter](http://msdn.microsoft.com/en-us/library/system.windows.controls.gridsplitter.aspx), [GroupBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.groupbox.aspx), [Panel](http://msdn.microsoft.com/en-us/library/system.windows.controls.panel.aspx), [ResizeGrip](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.resizegrip.aspx), [Separator](http://msdn.microsoft.com/en-us/library/system.windows.controls.separator.aspx), [ScrollBar](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.scrollbar.aspx), [ScrollViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.scrollviewer.aspx), [StackPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.stackpanel.aspx), [Thumb](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.thumb.aspx), [Viewbox](http://msdn.microsoft.com/en-us/library/system.windows.controls.viewbox.aspx), [VirtualizingStackPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingstackpanel.aspx), [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx), and [WrapPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.wrappanel.aspx).
* Media: [Image](http://msdn.microsoft.com/en-us/library/system.windows.controls.image.aspx), [MediaElement](http://msdn.microsoft.com/en-us/library/system.windows.controls.mediaelement.aspx), and [SoundPlayerAction](http://msdn.microsoft.com/en-us/library/system.windows.controls.soundplayeraction.aspx).
* Menus: [ContextMenu](http://msdn.microsoft.com/en-us/library/system.windows.controls.contextmenu.aspx), [Menu](http://msdn.microsoft.com/en-us/library/system.windows.controls.menu.aspx), and [ToolBar](http://msdn.microsoft.com/en-us/library/system.windows.controls.toolbar.aspx).
* Navigation: [Frame](http://msdn.microsoft.com/en-us/library/system.windows.controls.frame.aspx), [Hyperlink](http://msdn.microsoft.com/en-us/library/system.windows.documents.hyperlink.aspx), [Page](http://msdn.microsoft.com/en-us/library/system.windows.controls.page.aspx), [NavigationWindow](http://msdn.microsoft.com/en-us/library/system.windows.navigation.navigationwindow.aspx), and [TabControl](http://msdn.microsoft.com/en-us/library/system.windows.controls.tabcontrol.aspx).
* Selection: [CheckBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.checkbox.aspx), [ComboBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.combobox.aspx), [ListBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.listbox.aspx), [TreeView](http://msdn.microsoft.com/en-us/library/system.windows.controls.treeview.aspx), and [RadioButton](http://msdn.microsoft.com/en-us/library/system.windows.controls.radiobutton.aspx), [Slider](http://msdn.microsoft.com/en-us/library/system.windows.controls.slider.aspx).
* User Information: [AccessText](http://msdn.microsoft.com/en-us/library/system.windows.controls.accesstext.aspx), [Label](http://msdn.microsoft.com/en-us/library/system.windows.controls.label.aspx), [Popup](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.popup.aspx), [ProgressBar](http://msdn.microsoft.com/en-us/library/system.windows.controls.progressbar.aspx), [StatusBar](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.statusbar.aspx), [TextBlock](http://msdn.microsoft.com/en-us/library/system.windows.controls.textblock.aspx), and [ToolTip](http://msdn.microsoft.com/en-us/library/system.windows.controls.tooltip.aspx).

 Input and Commanding

Controls most often detect and respond to user input. The WPF input system uses both direct and routed events to support text input, focus management, and mouse positioning. For more information, see [Input Overview](http://msdn.microsoft.com/en-us/library/ms754010.aspx).

Applications often have complex input requirements. WPF provides a command system that separates user input actions from the code that responds to those actions. For more information, see [Commanding Overview](http://msdn.microsoft.com/en-us/library/ms752308.aspx).

 Layout

When you create a UI, you arrange your controls by location and size to form a layout. A key requirement of any layout is to adapt to changes in window size and display settings. Rather than forcing you to write the code to adapt a layout in these circumstances, WPF provides a first-class, extensible layout system for you.

The cornerstone of the layout system is relative positioning, which increases the ability to adapt to changing window and display conditions. In addition, the layout system manages the negotiation between controls to determine the layout. The negotiation is a two-step process: first, a control tells its parent what location and size it requires; second, the parent tells the control what space it can have.

The layout system is exposed to child controls through base WPF classes. For common layouts such as grids, stacking, and docking, WPF includes several layout controls:

* [Canvas](http://msdn.microsoft.com/en-us/library/system.windows.controls.canvas.aspx): Child controls provide their own layout.
* [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx): Child controls are aligned to the edges of the panel.
* [Grid](http://msdn.microsoft.com/en-us/library/system.windows.controls.grid.aspx): Child controls are positioned by rows and columns.
* [StackPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.stackpanel.aspx): Child controls are stacked either vertically or horizontally.
* [VirtualizingStackPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingstackpanel.aspx): Child controls are virtualized and arranged on a single line that is either horizontally or vertically oriented.
* [WrapPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.wrappanel.aspx): Child controls are positioned in left-to-right order and wrapped to the next line when there are more controls on the current line than space allows.

The following example uses a [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx) to lay out several [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) controls.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl283CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl283CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.LayoutWindow"

Title="Layout with the DockPanel" Height="143" Width="319">

<!--DockPanel to layout four text boxes-->

<DockPanel>

<TextBox DockPanel.Dock="Top">Dock = "Top"</TextBox>

<TextBox DockPanel.Dock="Bottom">Dock = "Bottom"</TextBox>

<TextBox DockPanel.Dock="Left">Dock = "Left"</TextBox>

<TextBox Background="White">This TextBox "fills" the remaining space.</TextBox>

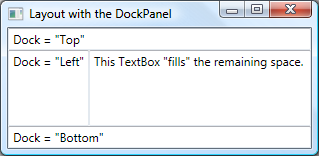
</DockPanel>

</Window>

The [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx) allows the child [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) controls to tell it how to arrange them. To do this, the [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx) implements a [Dock](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.dock.aspx) property that is exposed to the child controls to allow each of them to specify a dock style.

|  |
| --- |
| **Note:** |
| A property that is implemented by a parent control for use by child controls is a WPF construct called an attached property (see [Attached Properties Overview](http://msdn.microsoft.com/en-us/library/ms749011.aspx)). |

The following figure shows the result of the XAML markup in the preceding example.



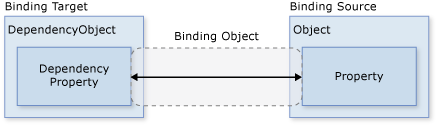
For more information, see [The Layout System](http://msdn.microsoft.com/en-us/library/ms745058.aspx). For an introductory sample, see [WPF Layout Gallery Sample](http://msdn.microsoft.com/en-us/library/ms771313.aspx).

 Data Binding

Most applications are created to provide users with the means to view and edit data. For WPF applications, the work of storing and accessing data is already provided for by technologies such as Microsoft SQL Server and ADO.NET. After the data is accessed and loaded into an application's managed objects, the hard work for WPF applications begins. Essentially, this involves two things:

1. Copying the data from the managed objects into controls, where the data can be displayed and edited.
2. Ensuring that changes made to data by using controls are copied back to the managed objects.

To simplify application development, WPF provides a data binding engine to automatically perform these steps. The core unit of the data binding engine is the [Binding](http://msdn.microsoft.com/en-us/library/system.windows.data.binding.aspx) class, whose job is to bind a control (the binding target) to a data object (the binding source). This relationship is illustrated by the following figure.



The following example demonstrates how to bind a [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) to an instance of a custom Person object. The Person implementation is shown in the following code.

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl313VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl313VisualBasic');)

Namespace SDKSample

Class Person

Private \_name As String = "No Name"

Public Property Name() As String

Get

Return \_name

End Get

Set(ByVal value As String)

\_name = value

End Set

End Property

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl314CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl314CSharp');)

namespace SDKSample

{

class Person

{

string name = "No Name";

public string Name

{

get { return name; }

set { name = value; }

}

}

}

The following markup binds the [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) to an instance of a custom Person object.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl316CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl316CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.DataBindingWindow">

...

<!-- Bind the TextBox to the data source (TextBox.Text to Person.Name) -->

<TextBox Name="personNameTextBox" Text="{Binding Path=Name}" />

...

</Window>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl317VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl317VisualBasic');)

Imports System.Windows ' Window

Namespace SDKSample

Partial Public Class DataBindingWindow

Inherits Window

Public Sub New()

InitializeComponent()

' Create Person data source

Dim person As Person = New Person()

' Make data source available for binding

Me.DataContext = person

End Sub

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl318CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl318CSharp');)

using System.Windows; // Window

namespace SDKSample

{

public partial class DataBindingWindow : Window

{

public DataBindingWindow()

{

InitializeComponent();

// Create Person data source

Person person = new Person();

// Make data source available for binding

this.DataContext = person;

}

}

}

In this example, the Person class is instantiated in code-behind and is set as the data context for the DataBindingWindow. In markup, the [Text](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.text.aspx) property of the [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) is bound to the Person.Name property (using the "{Binding ... }" XAML syntax). This XAML tells WPF to bind the [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) control to the Person object that is stored in the [DataContext](http://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext.aspx) property of the window.

The WPF data binding engine provides additional support that includes validation, sorting, filtering, and grouping. Furthermore, data binding supports the use of data templates to create custom UI for bound data when the UI displayed by the standard WPF controls is not appropriate.

For more information, see [Data Binding Overview](http://msdn.microsoft.com/en-us/library/ms752347.aspx). For an introductory sample, see [Data Binding Demo](http://msdn.microsoft.com/en-us/library/ms771319.aspx).

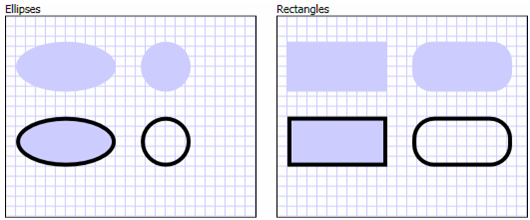
 Graphics

WPF introduces an extensive, scalable, and flexible set of graphics features that have the following benefits:

* Resolution-independent and device-independent graphics. The basic unit of measurement in the WPF graphics system is the device independent pixel, which is 1/96th of an inch, regardless of actual screen resolution, and provides the foundation for resolution-independent and device-independent rendering. Each device-independent pixel automatically scales to match the dots-per-inch (dpi) setting of the system it renders on.
* Improved precision. The WPF coordinate system is measured with double-precision floating-point numbers rather than single-precision. Transformations and opacity values are also expressed as double-precision. WPF also supports a wide color gamut (scRGB) and provides integrated support for managing inputs from different color spaces.
* Advanced graphics and animation support. WPF simplifies graphics programming by managing animation scenes for you; there is no need to worry about scene processing, rendering loops, and bilinear interpolation. Additionally, WPF provides hit-testing support and full alpha-compositing support.
* Hardware acceleration. The WPF graphics system takes advantage of graphics hardware to minimize CPU usage.

### 2-D Shapes

WPF provides a library of common vector-drawn 2-D shapes, such as the rectangles and ellipses that are shown in the following illustration.



An interesting capability of shapes is that they are not just for display; shapes implement many of the features that you expect from controls, including keyboard and mouse input. The following example shows the [MouseUp](http://msdn.microsoft.com/en-us/library/system.windows.uielement.mouseup.aspx) event of an [Ellipse](http://msdn.microsoft.com/en-us/library/system.windows.shapes.ellipse.aspx) being handled.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl350CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl350CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.EllipseEventHandlingWindow"

Title="Click the Ellipse">

<Ellipse Name="clickableEllipse" Fill="Blue" MouseUp="clickableEllipse\_MouseUp" />

</Window>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl351VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl351VisualBasic');)

Imports System.Windows ' Window, MessageBox

Imports System.Windows.Input ' MouseButtonEventArgs

Namespace SDKSample

Public Class EllipseEventHandlingWindow

Inherits Window

Public Sub New()

InitializeComponent()

End Sub

Private Sub clickableEllipse\_MouseUp(ByVal sender As Object, ByVal e As MouseButtonEventArgs)

MessageBox.Show("You clicked the ellipse!")

End Sub

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl352CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl352CSharp');)

using System.Windows; // Window, MessageBox

using System.Windows.Input; // MouseButtonEventHandler

namespace SDKSample

{

public partial class EllipseEventHandlingWindow : Window

{

public EllipseEventHandlingWindow()

{

InitializeComponent();

}

void clickableEllipse\_MouseUp(object sender, MouseButtonEventArgs e)

{

// Display a message

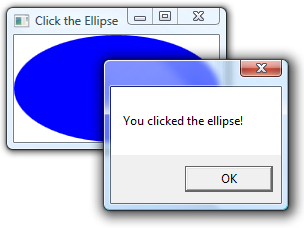
MessageBox.Show("You clicked the ellipse!");

}

}

}

The following figure shows what is produced by the preceding code.



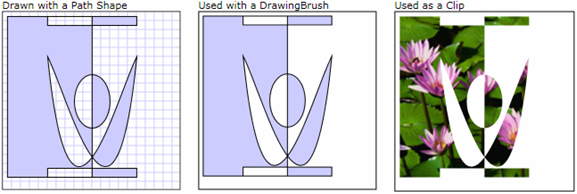
For more information, see [Shapes and Basic Drawing in WPF Overview](http://msdn.microsoft.com/en-us/library/ms747393.aspx). For an introductory sample, see [Shape Elements Sample](http://msdn.microsoft.com/en-us/library/ms771601.aspx).

### 2-D Geometries

The 2-D shapes provided by WPF cover the standard set of basic shapes. However, you may need to create custom shapes to facilitate the design of a customized UI. For this purpose, WPF provides geometries. The following figure demonstrates the use of geometries to create a custom shape that can be drawn directly, used as a brush, or used to clip other shapes and controls.

[Path](http://msdn.microsoft.com/en-us/library/system.windows.shapes.path.aspx) objects can be used to draw closed or open shapes, multiple shapes, and even curved shapes.

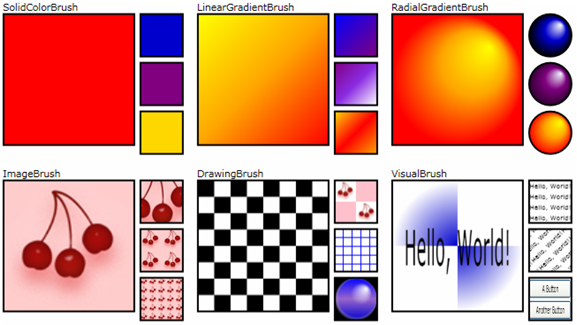
[Geometry](http://msdn.microsoft.com/en-us/library/system.windows.media.geometry.aspx) objects can be used for clipping, hit-testing, and rendering 2-D graphic data.



For more information, see [Geometry Overview](http://msdn.microsoft.com/en-us/library/ms751808.aspx). For an introductory sample, see [Geometries Sample](http://msdn.microsoft.com/en-us/library/ms771505.aspx).

### 2-D Effects

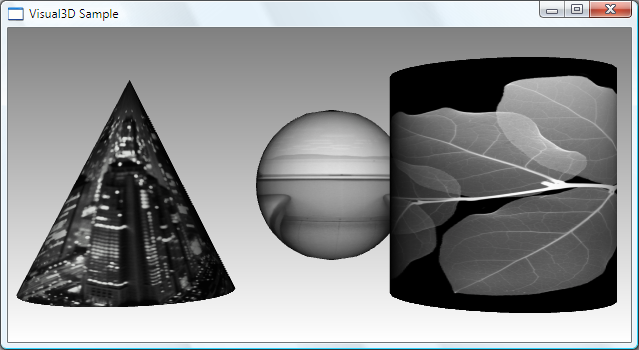
A subset of WPF 2-D capabilities includes visual effects, such as gradients, bitmaps, drawings, painting with videos, rotation, scaling, and skewing. These are all achieved with brushes; the following figure shows some examples.



For more information, see [WPF Brushes Overview](http://msdn.microsoft.com/en-us/library/aa970904.aspx). For an introductory sample, see [Brushes Sample](http://msdn.microsoft.com/en-us/library/ms771453.aspx).

### 3-D Rendering

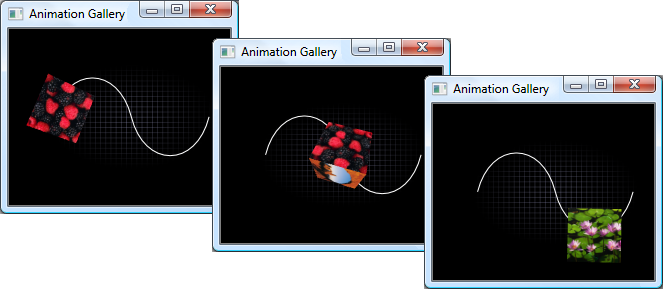
WPF also includes of 3-D rendering capabilities that integrate with 2-D graphics to allow the creation of more exciting and interesting UIs. For example, the following figure shows 2-D images rendered onto 3-D shapes.



For more information, see [3-D Graphics Overview](http://msdn.microsoft.com/en-us/library/ms747437.aspx). For an introductory sample, see [3-D Solids Sample](http://msdn.microsoft.com/en-us/library/ms771784.aspx).

 Animation

WPF animation support lets you make controls grow, shake, spin, and fade, to create interesting page transitions, and more. You can animate most WPF classes, even custom classes. The following figure shows a simple animation in action.



For more information, see [Animation Overview](http://msdn.microsoft.com/en-us/library/ms752312.aspx). For an introductory sample, see [Animation Example Gallery](http://msdn.microsoft.com/en-us/library/ms771741.aspx).

 Media

One way to convey rich content is through the use of audiovisual media. WPF provides special support for images, video, and audio.

### Images

Images are common to most applications, and WPF provides several ways to use them. The following figure shows a UI with a list box that contains thumbnail images. When a thumbnail is selected, the image is shown full-size.



For more information, see [Imaging Overview](http://msdn.microsoft.com/en-us/library/ms748873.aspx). For an introductory sample, see [WPF Photo Viewer Demo](http://msdn.microsoft.com/en-us/library/ms771331.aspx).

### Video and Audio

The [MediaElement](http://msdn.microsoft.com/en-us/library/system.windows.controls.mediaelement.aspx) control is capable of playing both video and audio, and it is flexible enough to be the basis for a custom media player. The following XAML markup implements a media player.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl390CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl390CSharp');)

<MediaElement

Name="myMediaElement"

Source="media/wpf.wmv"

LoadedBehavior="Manual"

Width="350" Height="250" />

The window in the following figure shows the [MediaElement](http://msdn.microsoft.com/en-us/library/system.windows.controls.mediaelement.aspx) control in action.



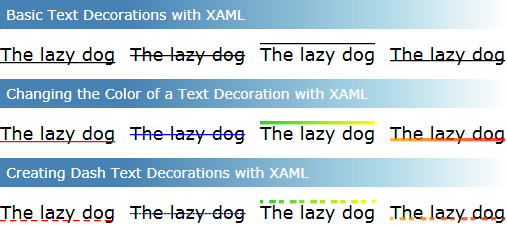
For more information, see [WPF Graphics, Animation, and Media Overview](http://msdn.microsoft.com/en-us/library/ms742562.aspx). For an introductory sample, see [Media Gallery](http://msdn.microsoft.com/en-us/library/ms771389.aspx).

 Text and Typography

To facilitate high-quality text rendering, WPF offers the following features:

* OpenType font support.
* ClearType enhancements.
* High performance that takes advantage of hardware acceleration.
* Integration of text with media, graphics, and animation.
* International font support and fallback mechanisms.

As a demonstration of text integration with graphics, the following figure shows the application of text decorations.



For more information, see [Typography in Windows Presentation Foundation](http://msdn.microsoft.com/en-us/library/ms742190.aspx). For introductory samples, see [Typography Samples](http://msdn.microsoft.com/en-us/library/ms771295.aspx).

 Documents

WPF has native support for working with three types of documents: flow documents, fixed documents, and XML Paper Specification (XPS) documents. WPF also provides the services to create, view, manage, annotate, package, and print documents.

### Flow Documents

Flow documents are designed to optimize viewing and readability by dynamically adjusting and reflowing content when window size and display settings change. The following XAML markup shows the definition of a [FlowDocument](http://msdn.microsoft.com/en-us/library/system.windows.documents.flowdocument.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl433CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl433CSharp');)

<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">

<Paragraph FontSize="18" FontWeight="Bold">Flow Document</Paragraph>

<Paragraph>

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy

nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi

enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis

nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.

</Paragraph>

...

</FlowDocument>

The following example demonstrates how to load a flow document into a [FlowDocumentReader](http://msdn.microsoft.com/en-us/library/system.windows.controls.flowdocumentreader.aspx) for viewing, searching, and printing.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl435CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl435CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.FlowDocumentReaderWindow"

Title="Flow Document Reader">

<FlowDocumentReader Name="flowDocumentReader" />

</Window>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl436VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl436VisualBasic');)

Imports System.Windows 'Window

Imports System.Windows.Documents 'FlowDocument

Imports System.IO 'FileStream, FileMode

Imports System.Windows.Markup 'XamlReader

Namespace SDKSample

Public Class FlowDocumentReaderWindow

Inherits Window

Public Sub New()

Me.InitializeComponent()

Using stream1 As FileStream = New FileStream("AFlowDocument.xaml", \_

FileMode.Open, FileAccess.Read)

Dim document1 As FlowDocument = \_

TryCast(XamlReader.Load(stream1), FlowDocument)

Me.flowDocumentReader.Document = document1

End Using

End Sub

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl437CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl437CSharp');)

using System.Windows; // Window

using System.Windows.Documents; // FlowDocument

using System.IO; // FileStream, FileMode

using System.Windows.Markup; // XamlReader

namespace SDKSample

{

public partial class FlowDocumentReaderWindow : System.Windows.Window

{

public FlowDocumentReaderWindow()

{

InitializeComponent();

// Open the file that contains the FlowDocument

using (FileStream xamlFile = new FileStream("AFlowDocument.xaml",

FileMode.Open, FileAccess.Read))

{

// Parse the file with the XamlReader.Load method

FlowDocument content = XamlReader.Load(xamlFile) as FlowDocument;

// Set the Document property to the parsed FlowDocument object

this.flowDocumentReader.Document = content;

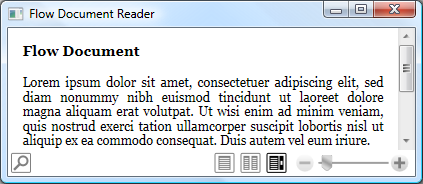
}

}

}

}

The following example shows the result.



For more information, see [Flow Document Overview](http://msdn.microsoft.com/en-us/library/aa970909.aspx). For an introductory sample, see [FlowDocument Properties Sample](http://msdn.microsoft.com/en-us/library/ms771316.aspx).

### Fixed Documents

Fixed documents are intended for applications that require a precise "what you see is what you get" (WYSIWYG) presentation, particularly with respect to printing. Typical uses for fixed documents include desktop publishing, word processing, and form layout, where adherence to the original page design is critical.

Fixed documents maintain the precise arrangement of their content in a device-independent manner. For example, a fixed document that is displayed on a 96 dots-per-inch (dpi) display appears the same as when it is printed to either a 600 dpi laser printer or a 4800 dpi photo typesetter. The layout remains the same in all cases, although the document's quality varies depending on the capabilities of each device.

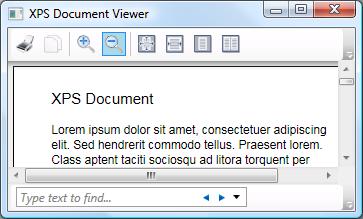
For more information, see [Documents in Windows Presentation Foundation](http://msdn.microsoft.com/en-us/library/ms748388.aspx).

### XPS Documents

XML Paper Specification (XPS) documents build on WPF's fixed documents. XPS documents are described with an XML-based schema that is essentially a paginated representation of electronic paper. XPS is an open, cross-platform document format that is designed to facilitate the creation, sharing, printing, and archiving of paginated documents. Important features of the XPS technology include the following:

* Packaging of XPS documents as [ZipPackage](http://msdn.microsoft.com/en-us/library/system.io.packaging.zippackage.aspx) files that conform to the [Open Packaging Conventions](http://www.microsoft.com/whdc/xps/xpspkg.mspx) (OPC).
* Hosting in both standalone and browser-based applications.
* Manual generation and manipulation of XPS documents from WPF applications.
* High-fidelity rendering by targeting maximum output device quality.
* Windows Vista print spooling.
* Direct routing of documents to XPS-compatible printers.
* UI integration with [DocumentViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx).

The following figure shows an XPS document that is displayed by a [DocumentViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx).

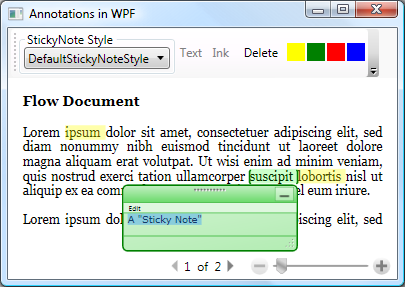


[DocumentViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx) also allows users to change the view, search, and print XPS documents.

For more information, see [Documents in Windows Presentation Foundation](http://msdn.microsoft.com/en-us/library/ms748388.aspx). For an introductory sample, see [Creating an XPS Document Sample](http://msdn.microsoft.com/en-us/library/ms771596.aspx).

### Annotations

Annotations are notes or comments that are added to documents to flag information or to highlight items of interest for later reference. Although writing notes on printed documents is easy, the ability to "write" notes on electronic documents is often limited or unavailable. In WPF, however, an annotations system is provided to support sticky notes and highlights. Additionally, these annotations can be applied to documents hosted in the [DocumentViewer](http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx) control, as shown in the following figure.



For more information, see [Annotations Overview](http://msdn.microsoft.com/en-us/library/ms748864.aspx). For an introductory sample, see [Document Serialization Sample](http://msdn.microsoft.com/en-us/library/ms771375.aspx).

### Packaging

The WPF [System.IO.Packaging](http://msdn.microsoft.com/en-us/library/system.io.packaging.aspx) APIs allow your applications to organize data, content, and resources into single, portable, easy-to-distribute, and easy-to-access ZIP documents. Digital signatures can be included to authenticate items that are contained in a package and to verify that the signed item was not tampered with or modified. You can also encrypt packages by using rights management in order to restrict access to protected information.

For more information, see [Documents in Windows Presentation Foundation](http://msdn.microsoft.com/en-us/library/ms748388.aspx). For samples, see [Packaging Samples](http://msdn.microsoft.com/en-us/library/ms771365.aspx).

### Printing

The .NET Framework includes a printing subsystem that WPF augments with support for enhanced print system control. Printing enhancements include the following:

* Real-time installation of remote print servers and queues.
* Dynamic discovery of printer capabilities.
* Dynamic setting of printer options.
* Print job rerouting and reprioritizing.

XPS documents also have a key performance enhancement. The existing Microsoft Windows Graphics Device Interface (GDI) print path typically requires two conversions:

* The first conversion of a document into a print processor format, such as Enhanced Metafile (EMF).
* A second conversion into the page description language of the printer, such as Printer Control Language (PCL) or PostScript.

However, XPS documents avoid these conversions because one component of the XPS file format is a print processor language and a page description language. This support helps to reduce both spool file size and networked printer loads.

For more information, see [Printing Overview](http://msdn.microsoft.com/en-us/library/ms742418.aspx). For an introductory sample, see [Printing an XPS Document](http://msdn.microsoft.com/en-us/library/ms771525.aspx).

 Customizing WPF Applications

Up to this point, you've seen the core WPF building blocks for developing applications. You use the application model to host and deliver application content, which consists mainly of controls. To simplify the arrangement of controls in a UI, and to ensure the arrangement is maintained in the face of changes to window size and display settings, you use the WPF layout system. Because most applications allow users to interact with data, you use data binding to reduce the work of integrating your UI with data. To enhance the visual appearance of your application, you use the comprehensive range of graphics, animation, and media support provided by WPF. Finally, if your application operates over text and documents, you can use the WPF text, typography, document, annotation, packaging, and printing capabilities.

Often, though, the basics are not enough for creating and managing a truly distinct and visually stunning user experience. The standard WPF controls may not integrate with the desired appearance of your application. Data may not be displayed in the most effective way. Your application's overall user experience may not be suited to the default look and feel of Windows themes. In many ways, a presentation technology needs visual extensibility as much as any other kind of extensibility.

For this reason, WPF provides a variety of mechanisms for creating unique user experiences, including a rich content model for controls, triggers, control and data templates, styles, UI resources, and themes and skins.

### Content Model

The main purpose of a majority of the WPF controls is to display content. In WPF, the type and number of items that can constitute the content of a control is referred to as the control's content model. Some controls can contain a single item and type of content; for example, the content of a [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx) is a string value that is assigned to the [Text](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.text.aspx) property. The following example sets the content of a [TextBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.textbox.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl559CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl559CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.TextBoxContentWindow"

Title="TextBox Content">

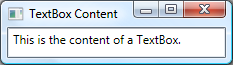
...

<TextBox Text="This is the content of a TextBox." />

...

</Window>

The following figure shows the result.



Other controls, however, can contain multiple items of different types of content; the content of a [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx), specified by the [Content](http://msdn.microsoft.com/en-us/library/system.windows.controls.contentcontrol.content.aspx) property, can contain a variety of items including layout controls, text, images, and shapes. The following example shows a [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) with content that includes a [DockPanel](http://msdn.microsoft.com/en-us/library/system.windows.controls.dockpanel.aspx), a [Label](http://msdn.microsoft.com/en-us/library/system.windows.controls.label.aspx), a [Border](http://msdn.microsoft.com/en-us/library/system.windows.controls.border.aspx), and a [MediaElement](http://msdn.microsoft.com/en-us/library/system.windows.controls.mediaelement.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl568CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl568CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.ButtonContentWindow"

Title="Button Content">

...

<Button Margin="20">

<!-- Button Content -->

<DockPanel Width="200" Height="180">

<Label DockPanel.Dock="Top" HorizontalAlignment="Center">Click Me!</Label>

<Border Background="Black" BorderBrush="Yellow" BorderThickness="2"

CornerRadius="2" Margin="5">

<MediaElement Source="media/wpf.wmv" Stretch="Fill" />

</Border>

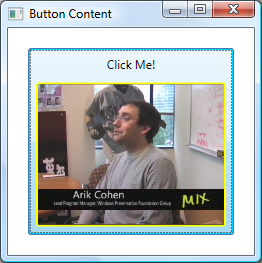
</DockPanel>

</Button>

...

</Window>

The following figure shows the content of this button.



For more information on the kinds of content that is supported by various controls, see [WPF Content Model](http://msdn.microsoft.com/en-us/library/bb613548.aspx).

### Triggers

Although the main purpose of XAML markup is to implement an application's appearance, you can also use XAML to implement some aspects of an application's behavior. One example is the use of triggers to change an application's appearance based on user interactions. For more information, see "Triggers" in [Styling and Templating](http://msdn.microsoft.com/en-us/library/ms745683.aspx).

### Control Templates

The default UIs for WPF controls are typically constructed from other controls and shapes. For example, a [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) is composed of both [ButtonChrome](http://msdn.microsoft.com/en-us/library/microsoft.windows.themes.buttonchrome.aspx) and [ContentPresenter](http://msdn.microsoft.com/en-us/library/system.windows.controls.contentpresenter.aspx) controls. The [ButtonChrome](http://msdn.microsoft.com/en-us/library/microsoft.windows.themes.buttonchrome.aspx) provides the standard button appearance, while the [ContentPresenter](http://msdn.microsoft.com/en-us/library/system.windows.controls.contentpresenter.aspx) displays the button's content, as specified by the [Content](http://msdn.microsoft.com/en-us/library/system.windows.controls.contentcontrol.content.aspx) property.

Sometimes the default appearance of a control may be incongruent with the overall appearance of an application. In this case, you can use a [ControlTemplate](http://msdn.microsoft.com/en-us/library/system.windows.controls.controltemplate.aspx) to change the appearance of the control's UI without changing its content and behavior.

For example, the following example shows how to change the appearance of a [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) by using a [ControlTemplate](http://msdn.microsoft.com/en-us/library/system.windows.controls.controltemplate.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl581CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl581CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.ControlTemplateButtonWindow"

Title="Button with Control Template" Height="158" Width="290">

<!-- Button using an ellipse -->

<Button Content="Click Me!" Click="button\_Click">

<Button.Template>

<ControlTemplate TargetType="{x:Type Button}">

<Grid Margin="5">

<Ellipse Stroke="DarkBlue" StrokeThickness="2">

<Ellipse.Fill>

<RadialGradientBrush Center="0.3,0.2" RadiusX="0.5" RadiusY="0.5">

<GradientStop Color="Azure" Offset="0.1" />

<GradientStop Color="CornflowerBlue" Offset="1.1" />

</RadialGradientBrush>

</Ellipse.Fill>

</Ellipse>

<ContentPresenter Name="content" HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Grid>

</ControlTemplate>

</Button.Template>

</Button>

</Window>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl582VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl582VisualBasic');)

Imports System.Windows ' Window, RoutedEventArgs, MessageBox

Namespace SDKSample

Public Class ControlTemplateButtonWindow

Inherits Window

Public Sub New()

InitializeComponent()

End Sub

Private Sub button\_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)

MessageBox.Show("Hello, Windows Presentation Foundation!")

End Sub

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl583CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl583CSharp');)

using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample

{

public partial class ControlTemplateButtonWindow : Window

{

public ControlTemplateButtonWindow()

{

InitializeComponent();

}

void button\_Click(object sender, RoutedEventArgs e)

{

// Show message box when button is clicked

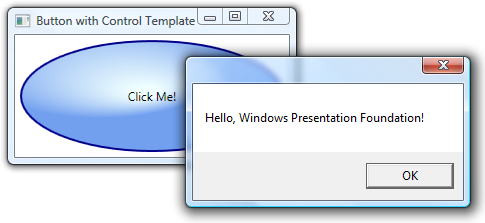
MessageBox.Show("Hello, Windows Presentation Foundation!");

}

}

}

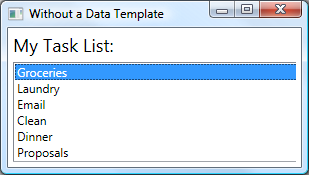
In this example, the default button UI has been replaced with an [Ellipse](http://msdn.microsoft.com/en-us/library/system.windows.shapes.ellipse.aspx) that has a dark blue border and is filled using a [RadialGradientBrush](http://msdn.microsoft.com/en-us/library/system.windows.media.radialgradientbrush.aspx). The [ContentPresenter](http://msdn.microsoft.com/en-us/library/system.windows.controls.contentpresenter.aspx) control displays the content of the [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx), "Click Me!" When the [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) is clicked, the [Click](http://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.buttonbase.click.aspx) event is still raised as part of the [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) control's default behavior. The result is shown in the following figure.



For more information, see [ControlTemplate](http://msdn.microsoft.com/en-us/library/system.windows.controls.controltemplate.aspx). For an introductory sample, see [Styling with ControlTemplates Sample](http://msdn.microsoft.com/en-us/library/ms771597.aspx).

### Data Templates

Whereas a control template lets you specify the appearance of a control, a data template lets you specify the appearance of a control's content. Data templates are frequently used to enhance how bound data is displayed. The following figure shows the default appearance for a [ListBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.listbox.aspx) that is bound to a collection of Task objects, where each task has a name, description, and priority.



The default appearance is what you would expect from a [ListBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.listbox.aspx). However, the default appearance of each task contains only the task name. To show the task name, description, and priority, the default appearance of the [ListBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.listbox.aspx) control's bound list items must be changed by using a [DataTemplate](http://msdn.microsoft.com/en-us/library/system.windows.datatemplate.aspx). The following XAML defines such a [DataTemplate](http://msdn.microsoft.com/en-us/library/system.windows.datatemplate.aspx), which is applied to each task by using the [ItemTemplate](http://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol.itemtemplate.aspx) attribute.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl601CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl601CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.DataTemplateWindow"

Title="With a Data Template">

...

<!-- Data Template (applied to each bound task item in the task collection) -->

<DataTemplate x:Key="myTaskTemplate">

<Border Name="border" BorderBrush="DarkSlateBlue" BorderThickness="2"

CornerRadius="2" Padding="5" Margin="5">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0" Padding="0,0,5,0" Text="Task Name:"/>

<TextBlock Grid.Row="0" Grid.Column="1" Text="{Binding Path=TaskName}"/>

<TextBlock Grid.Row="1" Grid.Column="0" Padding="0,0,5,0" Text="Description:"/>

<TextBlock Grid.Row="1" Grid.Column="1" Text="{Binding Path=Description}"/>

<TextBlock Grid.Row="2" Grid.Column="0" Padding="0,0,5,0" Text="Priority:"/>

<TextBlock Grid.Row="2" Grid.Column="1" Text="{Binding Path=Priority}"/>

</Grid>

</Border>

</DataTemplate>

...

<!-- Data template is specified by the ItemTemplate attribute -->

<ListBox

ItemsSource="{Binding}"

ItemTemplate="{StaticResource myTaskTemplate}"

HorizontalContentAlignment="Stretch"

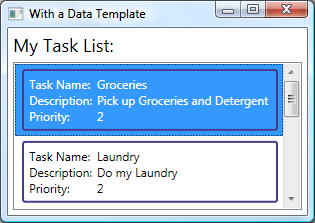
IsSynchronizedWithCurrentItem="True"

Margin="5,0,5,5" />

...

</Window>

The following figure shows the effect of this code.



Note that the [ListBox](http://msdn.microsoft.com/en-us/library/system.windows.controls.listbox.aspx) has retained its behavior and overall appearance; only the appearance of the content being displayed by the list box has changed.

For more information, see [Data Templating Overview](http://msdn.microsoft.com/en-us/library/ms742521.aspx). For an introductory sample, see [Introduction to Data Templating Sample](http://msdn.microsoft.com/en-us/library/aa972119.aspx).

### Styles

Styles enable developers and designers to standardize on a particular appearance for their product. WPF provides a strong style model, the foundation of which is the [Style](http://msdn.microsoft.com/en-us/library/system.windows.style.aspx) element. The following example creates a style that sets the background color for every [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) on a window to Orange.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl608CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl608CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.StyleWindow"

Title="Styles">

...

<!-- Style that will be applied to all buttons -->

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Orange" />

<Setter Property="BorderBrush" Value="Crimson" />

<Setter Property="FontSize" Value="20" />

<Setter Property="FontWeight" Value="Bold" />

<Setter Property="Margin" Value="5" />

</Style>

...

<!-- This button will have the style applied to it -->

<Button>Click Me!</Button>

<!-- This labe will not have the style applied to it -->

<Label>Don't Click Me!</Label>

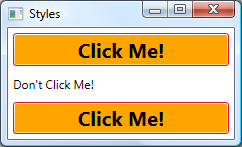
<!-- This button will have the style applied to it -->

<Button>Click Me!</Button>

...

</Window>

Because this style targets all [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) controls, the style is automatically applied to all the buttons in the window, as shown in the following figure.



For more information, see [Styling and Templating](http://msdn.microsoft.com/en-us/library/ms745683.aspx). For an introductory sample, see [Introduction to Styling and Templating Sample](http://msdn.microsoft.com/en-us/library/ms771659.aspx).

### Resources

Controls in an application should share the same appearance, which can include anything from fonts and background colors to control templates, data templates, and styles. You can use WPF's support for user interface (UI) resources to encapsulate these resources in a single location for reuse.

The following example defines a common background color that is shared by a [Button](http://msdn.microsoft.com/en-us/library/system.windows.controls.button.aspx) and a [Label](http://msdn.microsoft.com/en-us/library/system.windows.controls.label.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl615CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl615CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.ResourcesWindow"

Title="Resources Window">

<!-- Define window-scoped background color resource -->

<Window.Resources>

<SolidColorBrush x:Key="defaultBackground" Color="Red" />

</Window.Resources>

...

<!-- Button background is defined by window-scoped resource -->

<Button Background="{StaticResource defaultBackground}">One Button</Button>

<!-- Label background is defined by window-scoped resource -->

<Label Background="{StaticResource defaultBackground}">One Label</Label>

...

</Window>

This example implements a background color resource by using the Window.Resources property element. This resource is available to all children of the [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx). There are a variety of resource scopes, including the following, listed in the order in which they are resolved:

1. An individual control (using the inherited [FrameworkElement..::.Resources](http://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.resources.aspx) property).
2. A [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx) or a [Page](http://msdn.microsoft.com/en-us/library/system.windows.controls.page.aspx) (also using the inherited [FrameworkElement..::.Resources](http://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.resources.aspx) property).
3. An [Application](http://msdn.microsoft.com/en-us/library/system.windows.application.aspx) (using the [Application..::.Resources](http://msdn.microsoft.com/en-us/library/system.windows.application.resources.aspx) property).

The variety of scopes gives you flexibility with respect to the way in which you define and share your resources.

As an alternative to directly associating your resources with a particular scope, you can package one or more resources by using a separate [ResourceDictionary](http://msdn.microsoft.com/en-us/library/system.windows.resourcedictionary.aspx) that can be referenced in other parts of an application. For example, the following example defines a default background color in a resource dictionary.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl627CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl627CSharp');)

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<!-- Define background color resource -->

<SolidColorBrush x:Key="defaultBackground" Color="Red" />

<!-- Define other resources -->

...

</ResourceDictionary>

The following example references the resource dictionary defined in the previous example so that it is shared across an application.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl628CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl628CSharp');)

<Application

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.App">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="BackgroundColorResources.xaml"/>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

...

</Application>

Resources and resource dictionaries are the foundation of WPF support for themes and skins.

For more information, see [Resources Overview](http://msdn.microsoft.com/en-us/library/ms750613.aspx). For an introductory sample, see [Application Resources Sample](http://msdn.microsoft.com/en-us/library/ms771423.aspx).

### Themes and Skins

From a visual perspective, a theme defines the global appearance of Windows and the applications that run within it. Windows comes with several themes. For example, Microsoft Windows XP comes with both the Windows XP and Windows Classic themes, while Windows Vista comes with the Windows Vista and Windows Classic themes. The appearance that is defined by a theme defines the default appearance for a WPF application. WPF, however, does not integrate directly with Windows themes. Because the appearance of WPF is defined by templates, WPF includes one template for each of the well-known Windows themes, including Aero (Windows Vista), Classic (Microsoft Windows 2000), Luna (Microsoft Windows XP), and Royale (Microsoft Windows XP Media Center Edition 2005). These themes are packaged as resource dictionaries that are resolved if resources are not found in an application. Many applications rely on these themes to define their visual appearance; remaining consistent with Windows appearance helps users become familiar with more applications more easily.

On the other hand, the user experience for some applications does not necessarily come from the standard themes. For example, Microsoft Windows Media Player operates over audio and video data and benefits from a different style of user experience. Such UIs tend to provide customized, application-specific themes. These are known as skins, and applications that are skinned often provide hooks by which users can customize various aspects of the skin. Microsoft Windows Media Player has several prefabricated skins as well as a host of third-party skins.

Both themes and skins in WPF are most easily defined using resource dictionaries. The following example shows sample skin definitions.

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl631VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl631VisualBasic');)

<!-- Blue Skin -->

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDKSample">

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Blue" />

</Style>

...

</ResourceDictionary>

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl632CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl632CSharp');)

<!-- Blue Skin -->

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDKSample">

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Blue" />

</Style>

...

</ResourceDictionary>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl633VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl633VisualBasic');)

<!-- Yellow Skin -->

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDKSample">

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Yellow" />

</Style>

...

</ResourceDictionary>

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl634CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl634CSharp');)

<!-- Yellow Skin -->

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:SDKSample">

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Yellow" />

</Style>

...

</ResourceDictionary>

For more information, see "Shared Resources and Themes" in [Styling and Templating](http://msdn.microsoft.com/en-us/library/ms745683.aspx). For an introductory sample, see [Skinned Application Sample](http://msdn.microsoft.com/en-us/library/bb295250.aspx).

### Custom Controls

Although WPF provides a host of customization support, you may encounter situations where existing WPF controls do not meet the needs of either your application or its users. This can occur when:

* The UI that you require cannot be created by customizing the look and feel of existing WPF implementations.
* The behavior that you require is not supported (or not easily supported) by existing WPF implementations.

At this point, however, you can take advantage of one of three WPF models to create a new control. Each model targets a specific scenario and requires your custom control to derive from a particular WPF base class. The three models are listed here:

* User Control Model. A custom control derives from [UserControl](http://msdn.microsoft.com/en-us/library/system.windows.controls.usercontrol.aspx) and is composed of one or more other controls.
* Control Model. A custom control derives from [Control](http://msdn.microsoft.com/en-us/library/system.windows.controls.control.aspx) and is used to build implementations that separate their behavior from their appearance using templates, much like the majority of WPF controls. Deriving from [Control](http://msdn.microsoft.com/en-us/library/system.windows.controls.control.aspx) allows you more freedom for creating a custom UI than user controls, but it may require more effort.
* Framework Element Model. A custom control derives from [FrameworkElement](http://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.aspx) when its appearance is defined by custom rendering logic (not templates).

The following example shows a custom numeric up/down control that derives from [UserControl](http://msdn.microsoft.com/en-us/library/system.windows.controls.usercontrol.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl642CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl642CSharp');)

<UserControl

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.NumericUpDown">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<!-- Value text box -->

<Border BorderThickness="1" BorderBrush="Gray" Margin="2" Grid.RowSpan="2"

VerticalAlignment="Center" HorizontalAlignment="Stretch">

<TextBlock Name="valueText" Width="60" TextAlignment="Right" Padding="5"/>

</Border>

<!-- Up/Down buttons -->

<RepeatButton Name="upButton" Click="upButton\_Click" Grid.Column="1"

Grid.Row="0">Up</RepeatButton>

<RepeatButton Name="downButton" Click="downButton\_Click" Grid.Column="1"

Grid.Row="1">Down</RepeatButton>

</Grid>

</UserControl>

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl643VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl643VisualBasic');)

imports System 'EventArgs

imports System.Windows 'DependencyObject, DependencyPropertyChangedEventArgs,

' FrameworkPropertyMetadata, PropertyChangedCallback,

' RoutedPropertyChangedEventArgs

imports System.Windows.Controls 'UserControl

Namespace SDKSample

' Interaction logic for NumericUpDown.xaml

Partial Public Class NumericUpDown

Inherits System.Windows.Controls.UserControl

'NumericUpDown user control implementation

...

End Class

End Namespace

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl644CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl644CSharp');)

using System; // EventArgs

using System.Windows; // DependencyObject, DependencyPropertyChangedEventArgs,

// FrameworkPropertyMetadata, PropertyChangedCallback,

// RoutedPropertyChangedEventArgs

using System.Windows.Controls; // UserControl

namespace SDKSample

{

public partial class NumericUpDown : UserControl

{

// NumericUpDown user control implementation

...

}

}

The next example illustrates the XAML that is required to incorporate the user control into a [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx).

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl646CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl646CSharp');)

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Class="SDKSample.UserControlWindow"

xmlns:local="clr-namespace:SDKSample"

Title="User Control Window">

...

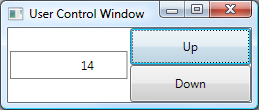
<!-- Numeric Up/Down user control -->

<local:NumericUpDown />

...

</Window>

The following figure shows the NumericUpDown control hosted in a [Window](http://msdn.microsoft.com/en-us/library/system.windows.window.aspx).



For more information on custom controls, see [Control Authoring Overview](http://msdn.microsoft.com/en-us/library/ms745025.aspx). For introductory samples, see [Control Customization Samples](http://msdn.microsoft.com/en-us/library/ms771268.aspx).

 WPF Best Practices

As with any development platform, WPF can be used in a variety of ways to achieve the desired result. As a way of ensuring that your WPF applications provide the required user experience and meet the demands of the audience in general, there are recommended best practices for accessibility, globalization and localization, and performance. See the following for more information:

* [Accessibility Best Practices](http://msdn.microsoft.com/en-us/library/aa350483.aspx)
* [WPF Globalization and Localization Overview](http://msdn.microsoft.com/en-us/library/ms788718.aspx)
* [Optimizing WPF Application Performance](http://msdn.microsoft.com/en-us/library/aa970683.aspx)
* [Windows Presentation Foundation Security](http://msdn.microsoft.com/en-us/library/aa970906.aspx)

 Summary

WPF is a comprehensive presentation technology for building a wide variety of visually stunning client applications. This introduction has provided a look at the key features of WPF.

The next step is to build WPF applications!

As you build them, you can come back to this introduction for a refresher on the key features and to find references to more detailed coverage of the features covered in this introduction.