Contents

Windows Presentation Foundation

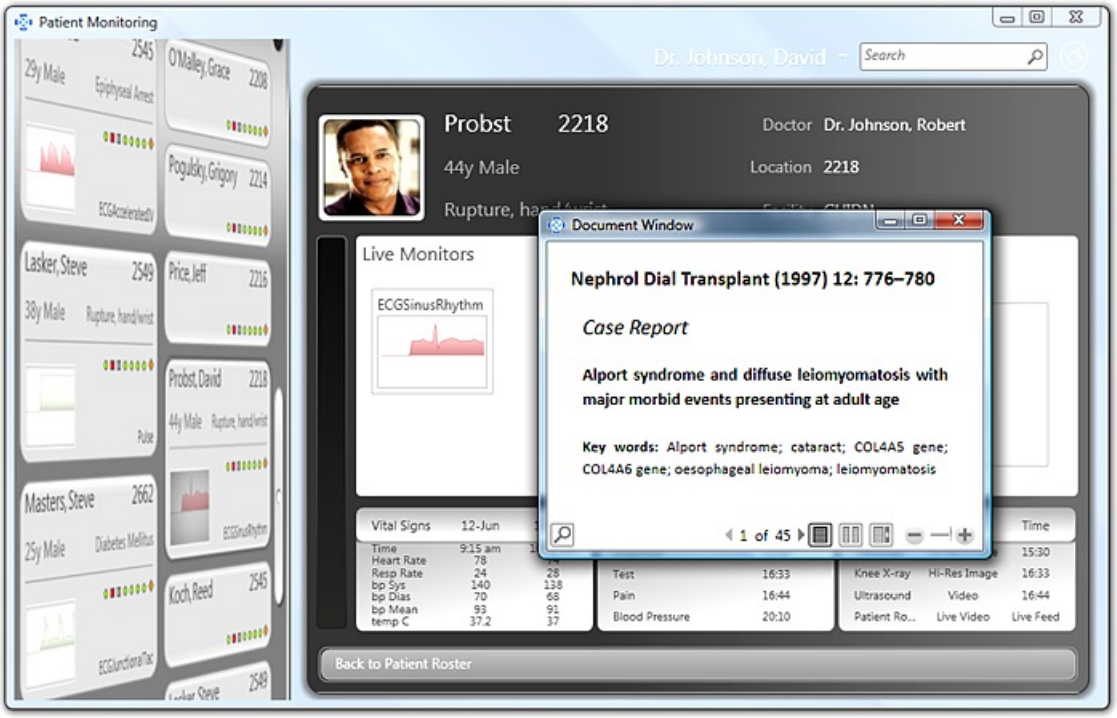
10/25/2019 • 2 minutes to read • Edit Online

Windows Presentation Foundation (WPF) provides developers with a unified programming model for building line-of-business desktop applications on Windows.

WPF overview

11/3/2019 • 22 minutes to read • Edit Online

Windows Presentation Foundation (WPF) lets you create desktop client applications for Windows with visually stunning user experiences.



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, 2D and 3D graphics, animation, styles, templates, documents, media, text, and typography. WPF is part of .NET, so you can build applications that incorporate other elements of the .NET API.

This overview is intended for newcomers and covers the key capabilities and concepts of WPF.

Program with WPF

WPF exists as a subset of .NET types that are (for the most part) located in the System.Windows namespace. If you have previously built applications with .NET 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, using your favorite .NET programming language, such as C# or Visual Basic.

WPF includes additional programming constructs that enhance properties and events: dependency properties and routed events.

Markup and code-behind

WPF lets you develop an application using both *markup* and *code-behind*, an experience with which ASP.NET developers should be familiar. You generally use 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.
- Globalization and localization for WPF applications is simplified.

Markup

XAML is an XML-based markup language that implements an application's appearance declaratively. You typically use it 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:

```
<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 class whose Title 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:

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.

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 implemented in code that is associated with markup. This type of code is known as code-behind. The following example shows the updated markup from the previous example and the code-behind:

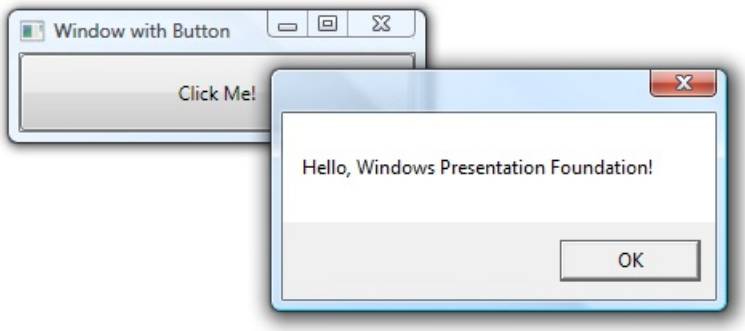
```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.AWindow"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>

</Window>
```

```
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!");
        }
    }
}
```

```
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
```

In this example, the code-behind implements a class that derives from the Window 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 event. When the button is clicked, the event handler shows a message box by calling the System.Windows.MessageBox.Show method.

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



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, and implement some behavior.

For more information, see Controls.

WPF controls by function

The built-in WPF controls are listed here:

- **Buttons**: Button and RepeatButton.
- **Data Display**: DataGrid, ListView, and TreeView.
- **Date Display and Selection**: Calendar and DatePicker.
- **Dialog Boxes**: OpenFileDialog, PrintDialog, and SaveFileDialog.
- **Digital Ink**: InkCanvas and InkPresenter.
- **Documents**: DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollViewer, and StickyNoteControl.
- **Input**: TextBox, RichTextBox, and PasswordBox.
- **Layout**: Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window, and WrapPanel.
- **Media**: Image, MediaElement, and SoundPlayerAction.
- **Menus**: ContextMenu, Menu, and ToolBar.
- **Navigation**: Frame, Hyperlink, Page, NavigationWindow, and TabControl.
- **Selection**: CheckBox, ComboBox, ListBox, RadioButton, and Slider.
- **User Information**: AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock, and ToolTip.

Input and commands

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.

Applications often have complex input requirements. WPF provides a command system that separates user-input actions from the code that responds to those actions.

Layout

When you create a user interface, 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: Child controls provide their own layout.
- DockPanel: Child controls are aligned to the edges of the panel.
- Grid: Child controls are positioned by rows and columns.
- StackPanel: Child controls are stacked either vertically or horizontally.
- VirtualizingStackPanel: Child controls are virtualized and arranged on a single line that is either horizontally or vertically oriented.
- WrapPanel: 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.

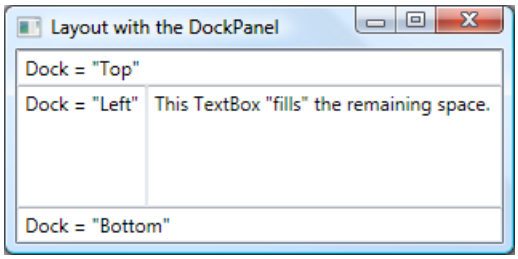
The following example uses a DockPanel to lay out several TextBox controls:

```
<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 allows the child TextBox controls to tell it how to arrange them. To do this, the DockPanel implements a `Dock` attached property that is exposed to the child controls to allow each of them to specify a dock style.

> **NOTE**
> A property that's implemented by a parent control for use by child controls is a WPF construct called an attached property.

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



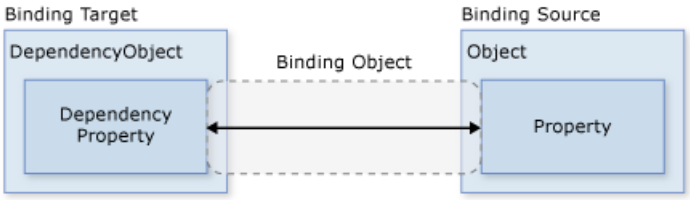
Data binding

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 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 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 next example demonstrates how to bind a TextBox to an instance of a custom `Person` object. The `Person` implementation is shown in the following code:

```
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
```

```
namespace SDKSample
{
    class Person
    {
        string name = "No Name";

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
}
```

The following markup binds the TextBox to an instance of a custom `Person` object:

```
 <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>
```

```
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
```

```
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 property of the TextBox is bound to the `Person.Name` property (using the "`{Binding ... }`" XAML syntax). This XAML tells WPF to bind the TextBox control to the `Person` object that is stored in the DataContext 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 user interface for bound data when the user interface displayed by the standard WPF controls is not appropriate.

For more information, see Data binding overview.

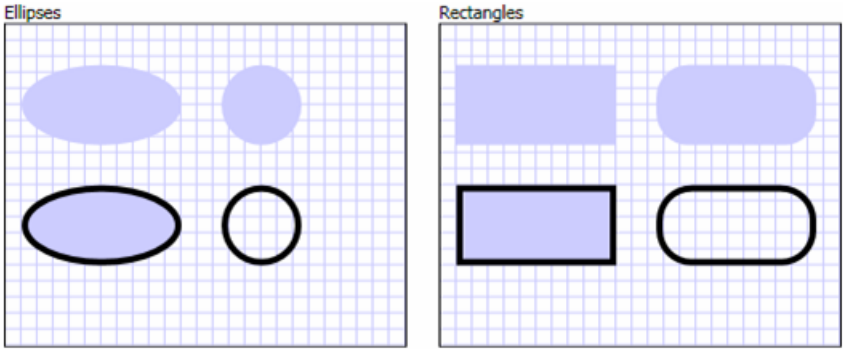
Graphics

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.

2D shapes

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



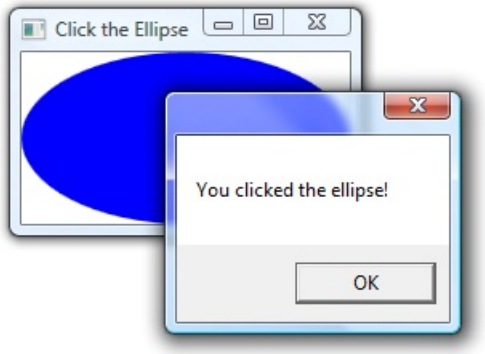
An interesting capability of shapes is that they are not just for display; shapes implement many of the features that you expect from controls, including keyboard and mouse input. The following example shows the MouseUp event of an Ellipse being handled:

```
<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>
```

```
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
```

```
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:



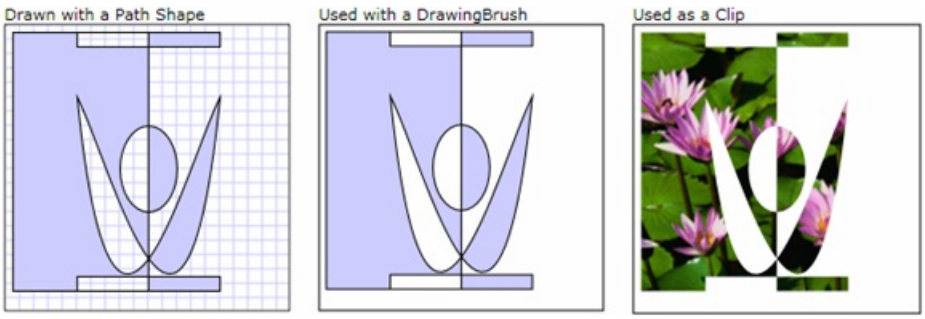
For more information, see Shapes and basic drawing in WPF overview.

2D geometries

The 2D 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 user interface. 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 objects can be used to draw closed or open shapes, multiple shapes, and even curved shapes.

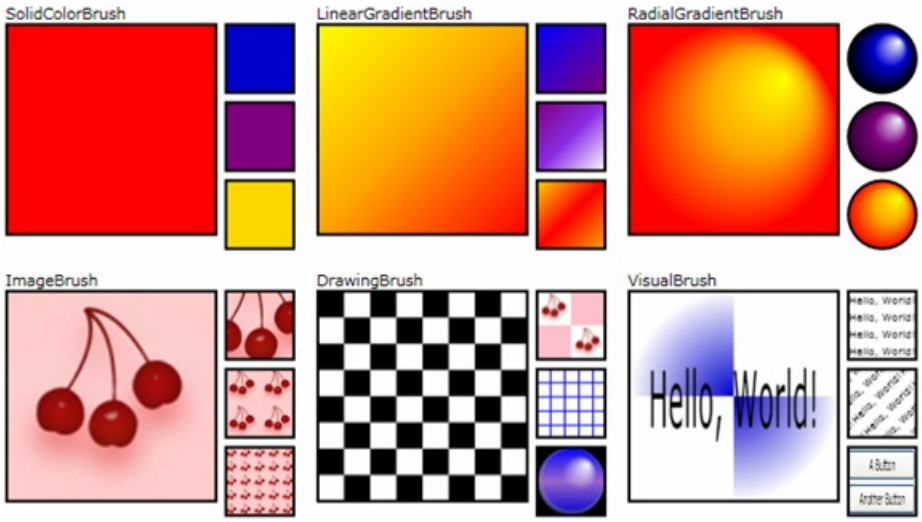
Geometry objects can be used for clipping, hit-testing, and rendering 2D graphic data.



For more information, see Geometry overview.

2D effects

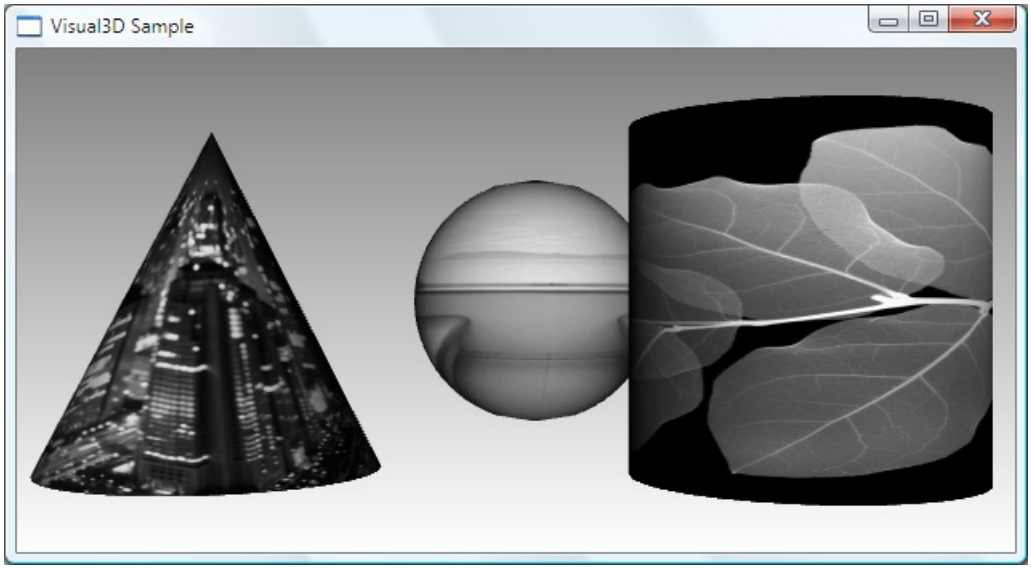
A subset of WPF 2D capabilities includes visual effects, such as gradients, bitmaps, drawings, painting with videos, rotation, scaling, and skewing. These are all achieved with brushes; the following figure shows some examples:



For more information, see WPF brushes overview.

3D rendering

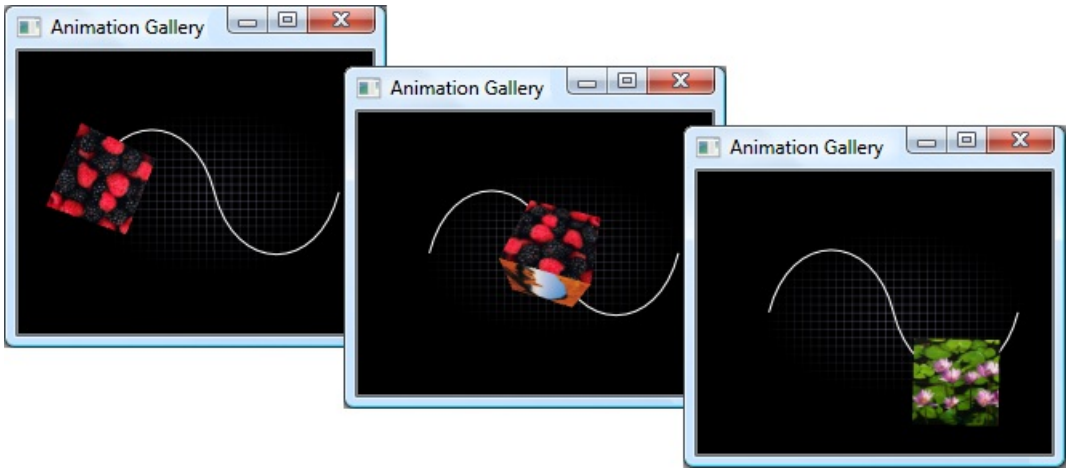
WPF also includes 3D rendering capabilities that integrate with 2-d graphics to allow the creation of more exciting and interesting user interfaces. For example, the following figure shows 2D images rendered onto 3D shapes:



For more information, see 3D graphics overview.

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.

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 user interface 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.

Video and audio

The MediaElement 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:

```
<MediaElement
  Name="myMediaElement"
  Source="media/wpf.wmv"
  LoadedBehavior="Manual"
  Width="350" Height="250" />
```

The window in the following figure shows the MediaElement control in action:

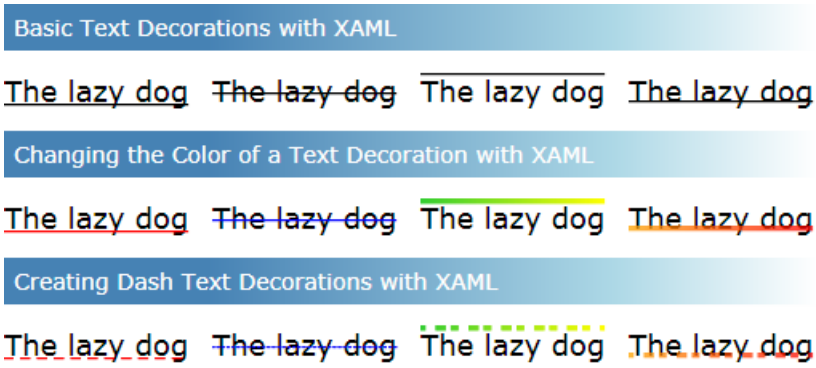
For more information, see Graphics and multimedia.

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.

Customize WPF apps

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 user interface, 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 let users interact with data, you use data binding to reduce the work of integrating your user interface with data. To enhance the visual appearance of your application, you use the comprehensive range of graphics, animation, and media support provided by WPF.

Often, though, the basics are not enough for creating and managing a truly distinct and visually stunning user experience. The standard WPF controls might not integrate with the desired appearance of your application. Data might 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 type of extensibility.

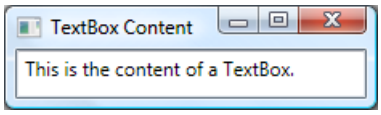
For this reason, WPF provides a variety of mechanisms for creating unique user experiences, including a rich content model for controls, triggers, control and data templates, styles, user interface resources, and themes and skins.

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 is a string value that is assigned to the Text property. The following example sets the content of a TextBox:

```
<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>
```

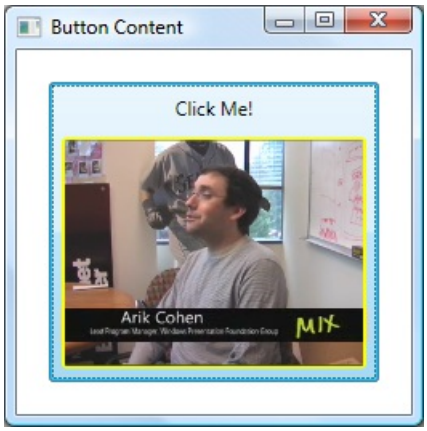
The following figure shows the result:



Other controls, however, can contain multiple items of different types of content; the content of a Button, specified by the Content property, can contain a variety of items including layout controls, text, images, and shapes. The following example shows a Button with content that includes a DockPanel, a Label, a Border, and a MediaElement:

```
<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.

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 Styles and templates.

Control templates

The default user interfaces for WPF controls are typically constructed from other controls and shapes. For example, a Button is composed of both ButtonChrome and ContentPresenter controls. The ButtonChrome provides the standard button appearance, while the ContentPresenter displays the button's content, as specified by the Content 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 to change the appearance of the control's user interface without changing its content and behavior.

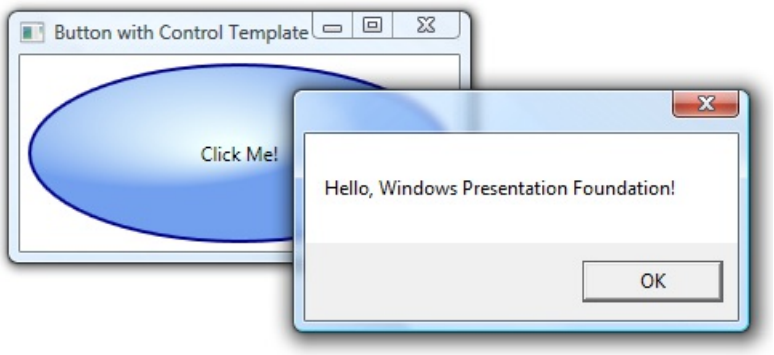
The following example shows how to change the appearance of a Button by using a ControlTemplate:

```
<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>
```

```
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!");
        }
    }
}
```

```
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
```

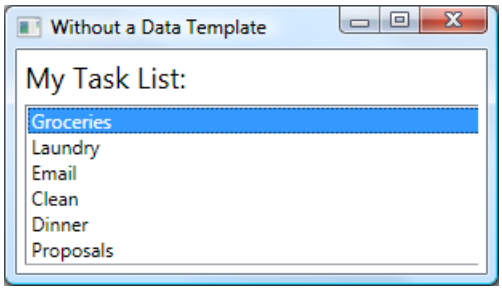
In this example, the default button user interface has been replaced with an Ellipse that has a dark blue border and is filled using a RadialGradientBrush. The ContentPresenter control displays the content of the Button, "Click Me!" When the Button is clicked, the Click event is still raised as part of the Button control's default behavior. The result is shown in the following figure:



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 that is bound to a collection of `Task` objects, where

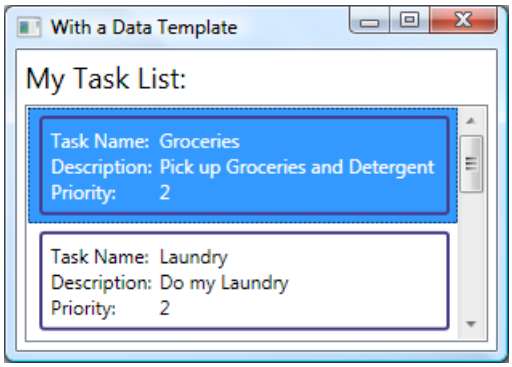
each task has a name, description, and priority:



The default appearance is what you would expect from a ListBox. 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 control's bound list items must be changed by using a DataTemplate. The following XAML defines such a DataTemplate, which is applied to each task by using the ItemTemplate attribute:

```
<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">
  <Window.Resources>
    <!-- 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>
  </Window.Resources>

  <!-- UI -->
  <DockPanel>
    <!-- Title -->
    <Label DockPanel.Dock="Top" FontSize="18" Margin="5" Content="My Task List:"/>

    <!-- Data template is specified by the ItemTemplate attribute -->
    <ListBox
      ItemsSource="{Binding}"
      ItemTemplate="{StaticResource myTaskTemplate}"
      HorizontalContentAlignment="Stretch"
      IsSynchronizedWithCurrentItem="True"
      Margin="5,0,5,5" />

  </DockPanel>
</Window>
```

The following figure shows the effect of this code:



Note that the ListBox has retained its behavior and overall appearance; only the appearance of the content being displayed by the list box has changed.

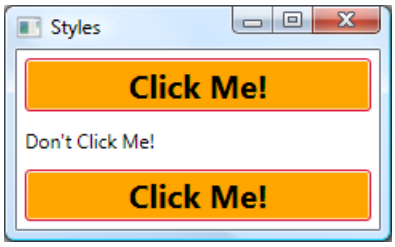
For more information, see Data templating overview.

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 element. The following example creates a style that sets the background color for every Button on a window to `Orange`:

```
<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 label 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 controls, the style is automatically applied to all the buttons in the window, as shown in the following figure:



For more information, see Styles and templates.

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

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 and a Label:

```
<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. 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 System.Windows.FrameworkElement.Resources property).
2. A Window or a Page (also using the inherited System.Windows.FrameworkElement.Resources property).
3. An Application (using the System.Windows.Application.Resources 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 that can be referenced in other parts of an application. For example, the following example defines a default background color in a resource dictionary:

```
<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:

```
<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.

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 user interface 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 and is composed of one or more other controls.
- **Control Model**. A custom control derives from Control 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 allows you more freedom for creating a custom user interface than user controls, but it may require more effort.
- **Framework Element Model**. A custom control derives from FrameworkElement 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:

```
<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>
```

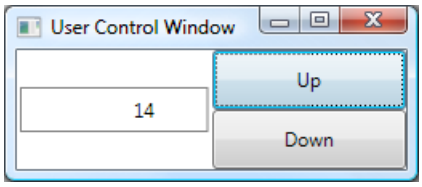
```
using System; // EventArgs
using System.Windows; // DependencyObject, DependencyPropertyChangedEventArgs,
                      // FrameworkPropertyMetadata, PropertyChangedCallback,
                      // RoutedPropertyChangedEventArgs
using System.Windows.Controls; // UserControl

namespace SDKSample
{
    public partial class NumericUpDown : UserControl
    {
        // NumericUpDown user control implementation
    }
}
```

```
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
```

The following example illustrates the XAML that is required to incorporate the user control into a Window:

```
<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:



For more information on custom controls, see Control authoring overview.

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. For more information, see:

- Accessibility
- WPF globalization and localization
- WPF app performance
- WPF security

Next steps

We've looked at the key features of WPF. Now it's time to build your first WPF app.

Walkthrough: My first WPF desktop app

See also

- Get started with WPF
- Windows Presentation Foundation
- WPF community resources

Getting Started (WPF)

11/3/2019 • 2 minutes to read • Edit Online

Windows Presentation Foundation (WPF) is a UI framework that creates desktop client applications. The WPF development platform supports a broad set of application development features, including an application model, resources, controls, graphics, layout, data binding, documents, and security. It is a subset of the .NET Framework, so if you have previously built applications with the .NET Framework using ASP.NET or Windows Forms, the programming experience should be familiar. WPF uses the Extensible Application Markup Language (XAML) to provide a declarative model for application programming. This section has topics that introduce and help you get started with WPF.

Where Should I Start?

I want to jump right in...	Walkthrough: My first WPF desktop application
How do I design the application UI?	Designing XAML in Visual Studio
New to .NET?	Overview of the .NET Framework .NET Framework Application Essentials Getting Started with Visual C# and Visual Basic
Tell me more about WPF...	Introduction to WPF in Visual Studio XAML Overview (WPF) Controls Data Binding Overview
Are you a Windows Forms developer?	Windows Forms Controls and Equivalent WPF Controls WPF and Windows Forms Interoperation

See also

- Class Library
- Application Development
- .NET Framework Developer Center

Application Development

11/1/2019 • 4 minutes to read • Edit Online

Windows Presentation Foundation (WPF) is a presentation framework that can be used to develop the following types of applications:

- Standalone Applications (traditional style Windows applications built as executable assemblies that are installed to and run from the client computer).
- XAML browser applications (XBAPs) (applications composed of navigation pages that are built as executable assemblies and hosted by Web browsers such as Microsoft Internet Explorer or Mozilla Firefox).
- Custom Control Libraries (non-executable assemblies containing reusable controls).
- Class Libraries (non-executable assemblies that contain reusable classes).

> **NOTE**
> Using WPF types in a Windows service is strongly discouraged. If you attempt to use these features in a Windows service, they may not work as expected.

To build this set of applications, WPF implements a host of services. This topic provides an overview of these services and where to find more information.

Application Management

Executable WPF applications commonly require a core set of functionality that includes the following:

- Creating and managing common application infrastructure (including creating an entry point method and a Windows message loop to receive system and input messages).
- Tracking and interacting with the lifetime of an application.
- Retrieving and processing command-line parameters.
- Sharing application-scope properties and UI resources.
- Detecting and processing unhandled exceptions.
- Returning exit codes.
- Managing windows in standalone applications.
- Tracking navigation in XAML browser applications (XBAPs), and standalone applications with navigation windows and frames.

These capabilities are implemented by the Application class, which you add to your applications using an *application definition*.

For more information, see Application Management Overview.

WPF Application Resource, Content, and Data Files

WPF extends the core support in the Microsoft .NET Framework for embedded resources with support for three kinds of non-executable data files: resource, content, and data. For more information, see WPF Application

Resource, Content, and Data Files.

A key component of the support for WPF non-executable data files is the ability to identify and load them using a unique URI. For more information, see Pack URIs in WPF.

Windows and Dialog Boxes

Users interact with WPF standalone applications through windows. The purpose of a window is to host application content and expose application functionality that usually allows users to interact with the content. In WPF, windows are encapsulated by the Window class, which supports:

- Creating and showing windows.
- Establishing owner/owned window relationships.
- Configuring window appearance (for example, size, location, icons, title bar text, border).
- Tracking and interacting with the lifetime of a window.

For more information, see WPF Windows Overview.

Window supports the ability to create a special type of window known as a dialog box. Both modal and modeless types of dialog boxes can be created.

For convenience, and the benefits of reusability and a consistent user experience across applications, WPF exposes three of the common Windows dialog boxes: OpenFileDialog, SaveFileDialog, and PrintDialog.

A message box is a special type of dialog box for showing important textual information to users, and for asking simple Yes/No/OK/Cancel questions. You use the MessageBox class to create and show message boxes.

For more information, see Dialog Boxes Overview.

Navigation

WPF supports Web-style navigation using pages (Page) and hyperlinks (Hyperlink). Navigation can be implemented in a variety of ways that include the following:

- Standalone pages that are hosted in a Web browser.
- Pages compiled into an XBAP that is hosted in a Web browser.
- Pages compiled into a standalone application and hosted by a navigation window (NavigationWindow).
- Pages that are hosted by a frame (Frame), which may be hosted in a standalone page, or a page compiled into either an XBAP or a standalone application.

To facilitate navigation, WPF implements the following:

- NavigationService, the shared navigation engine for processing navigation requests that is used by Frame, NavigationWindow, and XBAPs to support intra-application navigation.
- Navigation methods to initiate navigation.
- Navigation events to track and interact with navigation lifetime.
- Remembering back and forward navigation using a journal, which can also be inspected and manipulated.

For information, see Navigation Overview.

WPF also supports a special type of navigation known as structured navigation. Structured navigation can be used to call one or more pages that return data in a structured and predictable way that is consistent with calling

functions. This capability depends on the PageFunction<T> class, which is described further in Structured Navigation Overview. PageFunction<T> also serves to simplify the creation of complex navigation topologies, which are described in Navigation Topologies Overview.

Hosting

XBAPs can be hosted in Microsoft Internet Explorer or Firefox. Each hosting model has its own set of considerations and constraints that are covered in Hosting.

Build and Deploy

Although simple WPF applications can be built from a command prompt using command-line compilers, WPF integrates with Visual Studio to provide additional support that simplified the development and build process. For more information, see Building a WPF Application.

Depending on the type of application you build, there are one or more deployment options to choose from. For more information, see Deploying a WPF Application.

Related Topics

TITLE	DESCRIPTION
Application Management Overview	Provides an overview of the Application class including managing application lifetime, windows, application resources, and navigation.
Windows in WPF	Provides details of managing windows in your application including how to use the Window class and dialog boxes.
Navigation Overview	Provides an overview of managing navigation between pages of your application.
Hosting	Provides an overview of XAML browser applications (XBAPs).
Build and Deploy	Describes how to build and deploy your WPF application.
Introduction to WPF in Visual Studio	Describes the main features of WPF.
Walkthrough: My first WPF desktop application	A walkthrough that shows how to create a WPF application using page navigation, layout, controls, images, styles, and binding.

Advanced (Windows Presentation Foundation)

8/1/2019 • 2 minutes to read • Edit Online

This section describes some of the advanced areas in WPF.

In This Section

Controls

11/3/2019 • 7 minutes to read • Edit Online

Windows Presentation Foundation (WPF) ships with many of the common UI components that are used in almost every Windows application, such as Button, Label, TextBox, Menu, and ListBox. Historically, these objects have been referred to as controls. While the WPF SDK continues to use the term "control" to loosely mean any class that represents a visible object in an application, it is important to note that a class does not need to inherit from the Control class to have a visible presence. Classes that inherit from the Control class contain a ControlTemplate, which allows the consumer of a control to radically change the control's appearance without having to create a new subclass. This topic discusses how controls (both those that do inherit from the Control class and those that do not) are commonly used in WPF.

Creating an Instance of a Control

You can add a control to an application by using either Extensible Application Markup Language (XAML) or code. The following example shows how to create a simple application that asks a user for their first and last name. This example creates six controls: two labels, two text boxes, and two buttons, in XAML. All controls can be created similarly.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="30"/>
    <RowDefinition Height="30"/>
    <RowDefinition Height="30"/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

  <Label>
    Enter your first name:
  </Label>
  <TextBox Grid.Row="0" Grid.Column="1"
           Name="firstName" Margin="0,5,10,5"/>

  <Label Grid.Row="1" >
    Enter your last name:
  </Label>
  <TextBox Grid.Row="1" Grid.Column="1"
           Name="lastName" Margin="0,5,10,5"/>

  <Button Grid.Row="2" Grid.Column="0"
          Name="submit" Margin="2">
    View message
  </Button>

  <Button Grid.Row="2" Grid.Column="1"
          Name="Clear" Margin="2">
    Clear Name
  </Button>
</Grid>
```

The following example creates the same application in code. For brevity, the creation of the Grid, `grid1`, has been excluded from the sample. `grid1` has the same column and row definitions as shown in the preceding XAML

example.

```
Label firstNameLabel;
Label lastNameLabel;
TextBox firstName;
TextBox lastName;
Button submit;
Button clear;

void CreateControls()
{
    firstNameLabel = new Label();
    firstNameLabel.Content = "Enter your first name:";
    grid1.Children.Add(firstNameLabel);

    firstName = new TextBox();
    firstName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(firstName, 1);
    grid1.Children.Add(firstName);

    lastNameLabel = new Label();
    lastNameLabel.Content = "Enter your last name:";
    Grid.SetRow(lastNameLabel, 1);
    grid1.Children.Add(lastNameLabel);

    lastName = new TextBox();
    lastName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(lastName, 1);
    Grid.SetRow(lastName, 1);
    grid1.Children.Add(lastName);

    submit = new Button();
    submit.Content = "View message";
    Grid.SetRow(submit, 2);
    grid1.Children.Add(submit);

    clear = new Button();
    clear.Content = "Clear Name";
    Grid.SetRow(clear, 2);
    Grid.SetColumn(clear, 1);
    grid1.Children.Add(clear);

}
```

```
Private firstNameLabel As Label
Private lastNameLabel As Label
Private firstName As TextBox
Private lastName As TextBox
Private submit As Button
Private clear As Button

Sub CreateControls()
    firstNameLabel = New Label()
    firstNameLabel.Content = "Enter your first name:"
    grid1.Children.Add(firstNameLabel)

    firstName = New TextBox()
    firstName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(firstName, 1)
    grid1.Children.Add(firstName)

    lastNameLabel = New Label()
    lastNameLabel.Content = "Enter your last name:"
    Grid.SetRow(lastNameLabel, 1)
    grid1.Children.Add(lastNameLabel)

    lastName = New TextBox()
    lastName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(lastName, 1)
    Grid.SetRow(lastName, 1)
    grid1.Children.Add(lastName)

    submit = New Button()
    submit.Content = "View message"
    Grid.SetRow(submit, 2)
    grid1.Children.Add(submit)

    clear = New Button()
    clear.Content = "Clear Name"
    Grid.SetRow(clear, 2)
    Grid.SetColumn(clear, 1)
    grid1.Children.Add(clear)


End Sub
```

Changing the Appearance of a Control

It is common to change the appearance of a control to fit the look and feel of your application. You can change the appearance of a control by doing one of the following, depending on what you want to accomplish:

- Change the value of a property of the control.
- Create a Style for the control.
- Create a new ControlTemplate for the control.

Changing a Control's Property Value

Many controls have properties that allow you to change how the control appears, such as the Background of a Button. You can set the value properties in both XAML and code. The following example sets the Background, FontSize, and FontWeight properties on a Button in XAML.

```
<Button FontSize="14" FontWeight="Bold">
  <!--Set the Background property of the Button to
    a LinearGradientBrush.-->
  <Button.Background>
    <LinearGradientBrush StartPoint="0,0.5"
                         EndPoint="1,0.5">
      <GradientStop Color="Green" Offset="0.0" />
      <GradientStop Color="White" Offset="0.9" />
    </LinearGradientBrush>

  </Button.Background>
  View message
</Button>
```

The following example sets the same properties in code.

```
LinearGradientBrush buttonBrush = new LinearGradientBrush();
buttonBrush.StartPoint = new Point(0, 0.5);
buttonBrush.EndPoint = new Point(1, 0.5);
buttonBrush.GradientStops.Add(new GradientStop(Colors.Green, 0));
buttonBrush.GradientStops.Add(new GradientStop(Colors.White, 0.9));

submit.Background = buttonBrush;
submit.FontSize = 14;
submit.FontWeight = FontWeights.Bold;
```

```
Dim buttonBrush As New LinearGradientBrush()
buttonBrush.StartPoint = New Point(0, 0.5)
buttonBrush.EndPoint = New Point(1, 0.5)
buttonBrush.GradientStops.Add(New GradientStop(Colors.Green, 0))
buttonBrush.GradientStops.Add(New GradientStop(Colors.White, 0.9))

submit.Background = buttonBrush
submit.FontSize = 14
submit.FontWeight = FontWeights.Bold
```

Creating a Style for a Control

WPF gives you the ability to specify the appearance of controls wholesale, instead of setting properties on each instance in the application, by creating a Style. The following example creates a Style that is applied to each Button in the application. Style definitions are typically defined in XAML in a ResourceDictionary, such as the Resources property of the FrameworkElement.

```
<Style TargetType="Button">
  <Setter Property="FontSize" Value="14"/>
  <Setter Property="FontWeight" Value="Bold"/>
  <Setter Property="Background">
    <Setter.Value>
      <LinearGradientBrush StartPoint="0,0.5"
                           EndPoint="1,0.5">
        <GradientStop Color="Green" Offset="0.0" />
        <GradientStop Color="White" Offset="0.9" />
      </LinearGradientBrush>

    </Setter.Value>
  </Setter>
</Style>
```

You can also apply a style to only certain controls of a specific type by assigning a key to the style and specifying that key in the `Style` property of your control. For more information about styles, see Styling and Templating.

Creating a ControlTemplate

A Style allows you to set properties on multiple controls at a time, but sometimes you might want to customize the appearance of a Control beyond what you can do by creating a Style. Classes that inherit from the Control class have a ControlTemplate, which defines the structure and appearance of a Control. The Template property of a Control is public, so you can give a Control a ControlTemplate that is different than its default. You can often specify a new ControlTemplate for a Control instead of inheriting from a control to customize the appearance of a Control.

Consider the very common control, Button. The primary behavior of a Button is to enable an application to take some action when the user clicks it. By default, the Button in WPF appears as a raised rectangle. While developing an application, you might want to take advantage of the behavior of a Button--that is, by handling the button's click event--but you might change the button's appearance beyond what you can do by changing the button's properties. In this case, you can create a new ControlTemplate.

The following example creates a ControlTemplate for a Button. The ControlTemplate creates a Button with rounded corners and a gradient background. The ControlTemplate contains a Border whose Background is a LinearGradientBrush with two GradientStop objects. The first GradientStop uses data binding to bind the Color property of the GradientStop to the color of the button's background. When you set the Background property of the Button, the color of that value will be used as the first GradientStop. For more information about data binding, see Data Binding Overview. The example also creates a Trigger that changes the appearance of the Button when IsPressed is `true`.

```
<!--Define a template that creates a gradient-colored button.-->
<Style TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="Button">
        <Border
          x:Name="Border"
          CornerRadius="20"
          BorderThickness="1"
          BorderBrush="Black">
          <Border.Background>
            <LinearGradientBrush StartPoint="0,0.5"
                                 EndPoint="1,0.5">
              <GradientStop Color="{Binding Background.Color,
                    RelativeSource={RelativeSource TemplatedParent}}"
                            Offset="0.0" />
              <GradientStop Color="White" Offset="0.9" />
            </LinearGradientBrush>
          </Border.Background>
          <ContentPresenter
            Margin="2"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            RecognizesAccessKey="True"/>
        </Border>
        <ControlTemplate.Triggers>
          <!--Change the appearance of
          the button when the user clicks it.-->
          <Trigger Property="IsPressed" Value="true">
            <Setter TargetName="Border" Property="Background">
              <Setter.Value>
                <LinearGradientBrush StartPoint="0,0.5"
                                     EndPoint="1,0.5">
                  <GradientStop Color="{Binding Background.Color,
                    RelativeSource={RelativeSource TemplatedParent}}"
                                Offset="0.0" />
                  <GradientStop Color="DarkSlateGray" Offset="0.9" />
                </LinearGradientBrush>
              </Setter.Value>
            </Setter>
          </Trigger>

        </ControlTemplate.Triggers>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName"
        Background="Green">View message</Button>
```

> **NOTE**
>
> The Background property of the Button must be set to a SolidColorBrush for the example to work properly.

Subscribing to Events

You can subscribe to a control's event by using either XAML or code, but you can only handle an event in code. The following example shows how to subscribe to the `Click` event of a Button.

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName" Click="submit_Click"
  Background="Green">View message</Button>
```

```
submit.Click += new RoutedEventHandler(submit_Click);
```

```
AddHandler submit.Click, AddressOf submit_Click
```

The following example handles the `Click` event of a Button.

```
void submit_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text);
}
```

```
Private Sub submit_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text)

End Sub
```

Rich Content in Controls

Most classes that inherit from the Control class have the capacity to contain rich content. For example, a Label can contain any object, such as a string, an Image, or a Panel. The following classes provide support for rich content and act as base classes for most of the controls in WPF.

- ContentControl-- Some examples of classes that inherit from this class are Label, Button, and ToolTip.
- ItemsControl-- Some examples of classes that inherit from this class are ListBox, Menu, and StatusBar.
- HeaderedContentControl-- Some examples of classes that inherit from this class are TabItem, GroupBox, and Expander.
- HeaderedItemsControl--Some examples of classes that inherit from this class are MenuItem, TreeViewItem, and ToolBar.

For more information about these base classes, see WPF Content Model.

See also

- Styling and Templating
- Controls by Category
- Control Library
- Data Templating Overview
- Data Binding Overview
- Input
- Enable a Command
- Walkthroughs: Create a Custom Animated Button
- Control Customization

Data

11/3/2019 • 2 minutes to read • Edit Online

Windows Presentation Foundation (WPF) data binding provides a simple and consistent way for applications to present and interact with data. Elements can be bound to data from a variety of data sources in the form of common language runtime (CLR) objects and XML. Windows Presentation Foundation (WPF) also provides a mechanism for the transfer of data through drag-and-drop operations.

In This Section

Data Binding
Drag and Drop

Reference

System.Windows.Data

Binding

DataTemplate

DataTemplateSelector

Related Sections

Controls

Styling and Templating

Data Binding

See also

- Walkthrough: My first WPF desktop application
- Walkthrough: Caching Application Data in a WPF Application

Graphics and Multimedia

10/25/2019 • 4 minutes to read • Edit Online

Windows Presentation Foundation (WPF) provides support for multimedia, vector graphics, animation, and content composition, making it easy for developers to build interesting user interfaces and content. Using Visual Studio, you can create vector graphics or complex animations and integrate media into your applications.

This topic introduces the graphics, animation, and media features of WPF, which enable you to add graphics, transition effects, sound, and video to your applications.

> **NOTE**
> Using WPF types in a Windows service is strongly discouraged. If you attempt to use WPF types in a Windows service, the service may not work as expected.

What's New with Graphics and Multimedia in WPF 4

Several changes have been made related to graphics and animations.

- Layout Rounding

  When an object edge falls in the middle of a pixel device, the DPI-independent graphics system can create rendering artifacts, such as blurry or semi-transparent edges. Previous versions of WPF included pixel snapping to help handle this case. Silverlight 2 introduced layout rounding, which is another way to move elements so that edges fall on whole pixel boundaries. WPF now supports layout rounding with the UseLayoutRounding attached property on FrameworkElement.

- Cached Composition

  By using the new BitmapCache and BitmapCacheBrush classes, you can cache a complex part of the visual tree as a bitmap and greatly improve rendering time. The bitmap remains responsive to user input, such as mouse clicks, and you can paint it onto other elements just like any brush.

- Pixel Shader 3 Support

  WPF 4 builds on top of the ShaderEffect support introduced in WPF 3.5 SP1 by allowing applications to write effects by using Pixel Shader (PS) version 3.0. The PS 3.0 shader model is more sophisticated than PS 2.0, which allows for even more effects on supported hardware.

- Easing Functions

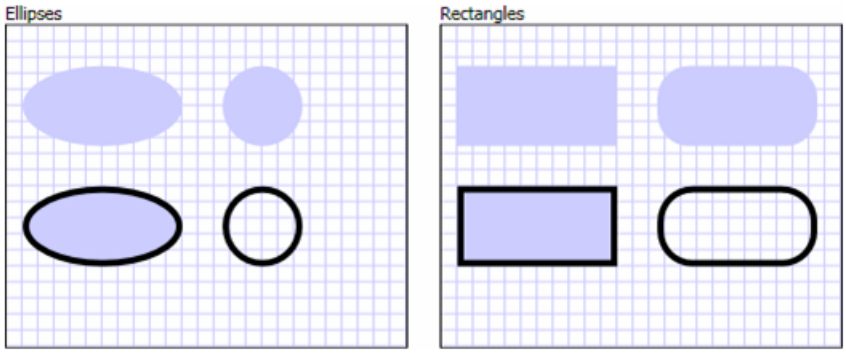
  You can enhance animations with easing functions, which give you additional control over the behavior of animations. For example, you can apply an ElasticEase to an animation to give the animation a springy behavior. For more information, see the easing types in the System.Windows.Media.Animation namespace.

Graphics and Rendering

WPF includes support for high quality 2-D graphics. The functionality includes brushes, geometries, images, shapes and transformations. For more information, see Graphics. The rendering of graphical elements is based on the Visual class. The structure of visual objects on the screen is described by the visual tree. For more information, see WPF Graphics Rendering Overview.

2-D Shapes

WPF provides a library of commonly used, vector-drawn 2-D shapes, such as rectangles and ellipses, which the following illustration shows.



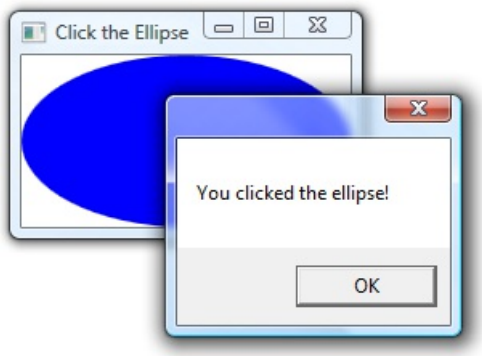
These intrinsic WPF shapes are not just shapes: they are programmable elements that implement many of the features that you expect from most common controls, which include keyboard and mouse input. The following example shows how to handle the MouseUp event raised by clicking an Ellipse element.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="Window1" >
  <Ellipse Fill="LightBlue" MouseUp="ellipseButton_MouseUp" />
</Window>
```

```
public partial class Window1  : Window
{
    void ellipseButton_MouseUp(object sender, MouseButtonEventArgs e)
    {
        MessageBox.Show("You clicked the ellipse!");
    }
}
```

```
Partial Public Class Window1
    Inherits Window
    Private Sub ellipseButton_MouseUp(ByVal sender As Object, ByVal e As MouseButtonEventArgs)
        MessageBox.Show("You clicked the ellipse!")
    End Sub
End Class
```

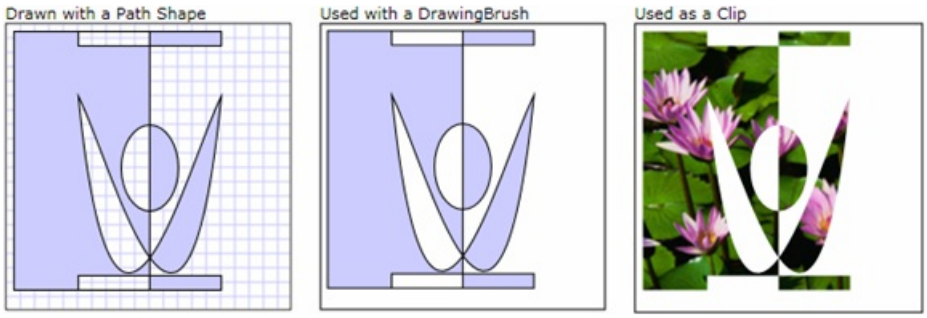
The following illustration shows the output for the preceding XAML markup and code-behind.



For more information, see Shapes and Basic Drawing in WPF Overview. For an introductory sample, see Shape Elements Sample.

2-D Geometries

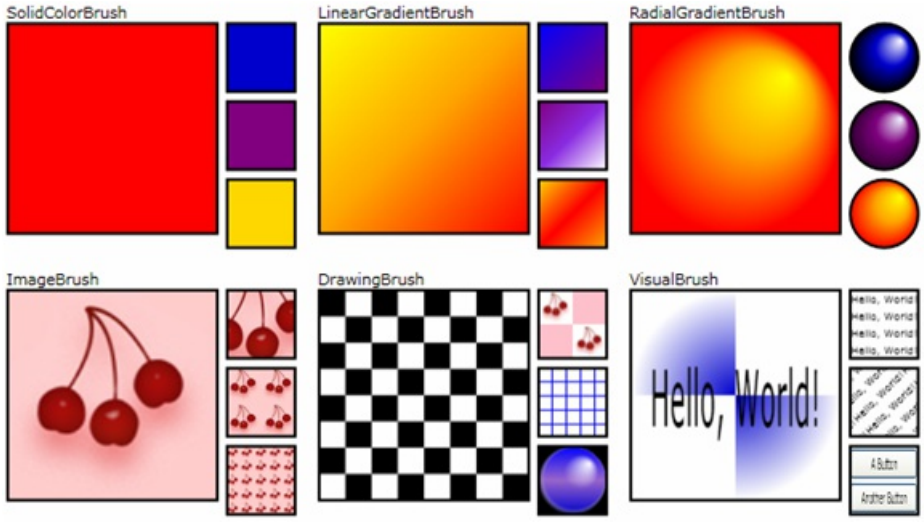
When the 2-D shapes that WPF provides are not sufficient, you can use WPF support for geometries and paths to create your own. The following illustration shows how you can use geometries to create shapes, as a drawing brush, and to clip other WPF elements.



For more information, see Geometry Overview. For an introductory sample, see Geometries Sample.

2-D Effects

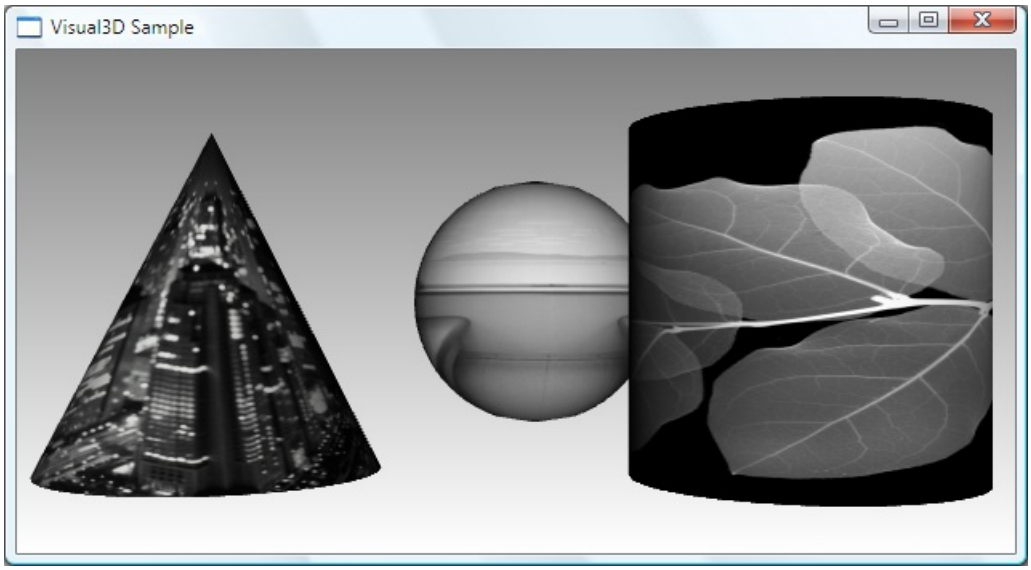
WPF provides a library of 2-D classes that you can use to create a variety of effects. The 2-D rendering capability of WPF provides the ability to paint UI elements that have gradients, bitmaps, drawings, and videos; and to manipulate them by using rotation, scaling, and skewing. The following illustration gives an example of the many effects you can achieve by using WPF brushes.



For more information, see WPF Brushes Overview. For an introductory sample, see Brushes Sample.

3-D Rendering

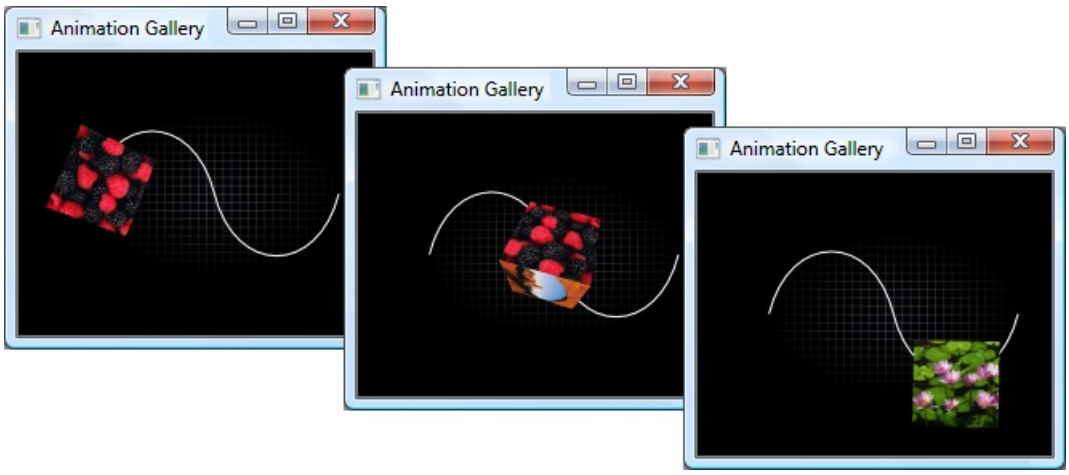
WPF provides a set of 3-D rendering capabilities that integrate with 2-D graphics support in WPF in order for you to create more exciting layout, UI, and data visualization. At one end of the spectrum, WPF enables you to render 2-D images onto the surfaces of 3-D shapes, which the following illustration demonstrates.



For more information, see 3-D Graphics Overview. For an introductory sample, see 3-D Solids Sample.

Animation

Use animation to make controls and elements grow, shake, spin, and fade; and to create interesting page transitions, and more. Because WPF enables you to animate most properties, not only can you animate most WPF objects, you can also use WPF to animate custom objects that you create.



For more information, see Animation Overview. For an introductory sample, see Animation Example Gallery.

Media

Images, video, and audio are media-rich ways of conveying information and user experiences.

Images

Images, which include icons, backgrounds, and even parts of animations, are a core part of most applications. Because you frequently need to use images, WPF exposes the ability to work with them in a variety of ways. The following illustration shows just one of those ways.

For more information, see Imaging Overview.

Video and Audio

A core feature of the graphics capabilities of WPF is to provide native support for working with multimedia, which includes video and audio. The following example shows how to insert a media player into an application.

```
<MediaElement Source="media\numbers.wmv" Width="450" Height="250" />
```

MediaElement is capable of playing both video and audio, and is extensible enough to allow the easy creation of custom UIs.

For more information, see the Multimedia Overview.

See also

- System.Windows.Media
- System.Windows.Media.Animation
- System.Windows.Media.Media3D
- 2D Graphics and Imaging
- Shapes and Basic Drawing in WPF Overview
- Painting with Solid Colors and Gradients Overview
- Painting with Images, Drawings, and Visuals
- Animation and Timing How-to Topics
- 3-D Graphics Overview
- Multimedia Overview

Security (WPF)

11/3/2019 • 11 minutes to read • Edit Online

When developing Windows Presentation Foundation (WPF) standalone and browser-hosted applications, you must consider the security model. WPF standalone applications execute with unrestricted permissions (CAS **FullTrust** permission set), whether deployed using Windows Installer (.msi), XCopy, or ClickOnce. Deploying partial-trust, standalone WPF applications with ClickOnce is unsupported. However, a full-trust host application can create a partial-trust AppDomain using the .NET Framework Add-in model. For more information, see WPF Add-Ins Overview.

WPF browser-hosted applications are hosted by Windows Internet Explorer or Firefox, and can be either XAML browser applications (XBAPs) or loose Extensible Application Markup Language (XAML) documents For more information, see WPF XAML Browser Applications Overview.

WPF browser-hosted applications execute within a partial trust security sandbox, by default, which is limited to the default CAS **Internet** zone permission set. This effectively isolates WPF browser-hosted applications from the client computer in the same way that you would expect typical Web applications to be isolated. An XBAP can elevate privileges, up to Full Trust, depending on the security zone of the deployment URL and the client's security configuration. For more information, see WPF Partial Trust Security.

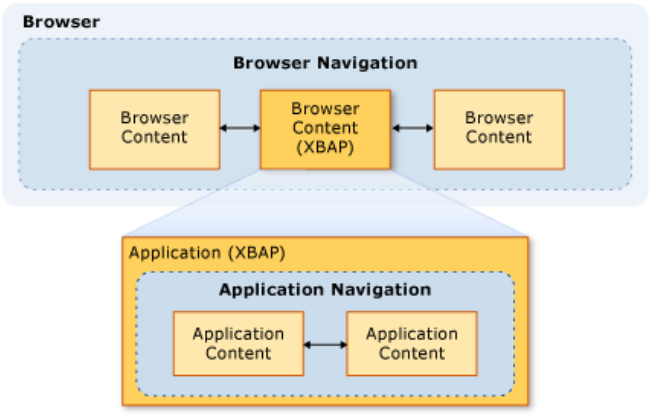
This topic discusses the security model for Windows Presentation Foundation (WPF) standalone and browser-hosted applications.

This topic contains the following sections:

Safe Navigation

For XBAPs, WPF distinguishes two types of navigation: application and browser.

Application navigation is navigation between items of content within an application that is hosted by a browser. *Browser navigation* is navigation that changes the content and location URL of a browser itself. The relationship between application navigation (typically XAML) and browser navigation (typically HTML) is shown in the following illustration:



The type of content that is considered safe for an XBAP to navigate to is primarily determined by whether application

navigation or browser navigation is used.

Application Navigation Security

Application navigation is considered safe if it can be identified with a pack URI, which supports four types of content:

CONTENT TYPE	DESCRIPTION	URI EXAMPLE
Resource	Files that are added to a project with a build type of **Resource**.	`pack://application:,,,/MyResourceFile.xaml`
Content	Files that are added to a project with a build type of **Content**.	`pack://application:,,,/MyContentFile.xaml`
Site of origin	Files that are added to a project with a build type of **None**.	`pack://siteoforigin:,,,/MySiteOfOriginFile.x`
Application code	XAML resources that have a compiled code-behind. -or- XAML files that are added to a project with a build type of **Page**.	`pack://application:,,,/MyResourceFile.xaml`

> **NOTE**
> For more information about application data files and pack URIs, see WPF Application Resource, Content, and Data Files.

Files of these content types can be navigated to by either the user or programmatically:

- **User Navigation**. The user navigates by clicking a Hyperlink element.
- **Programmatic Navigation**. The application navigates without involving the user, for example, by setting the NavigationWindow.Source property.

Browser Navigation Security

Browser navigation is considered safe only under the following conditions:

- **User Navigation**. The user navigates by clicking a Hyperlink element that is within the main NavigationWindow, not in a nested Frame.
- **Zone**. The content being navigated to is located on the Internet or the local intranet.
- **Protocol**. The protocol being used is either **http**, **https**, **file**, or **mailto**.

If an XBAP attempts to navigate to content in a manner that does not comply with these conditions, a SecurityException is thrown.

Web Browsing Software Security Settings

The security settings on your computer determine the access that any Web browsing software is granted. Web browsing software includes any application or component that uses the WinINet or UrlMon APIs, including Internet Explorer and PresentationHost.exe.

Internet Explorer provides a mechanism by which you can configure the functionality that is allowed to be executed by or from Internet Explorer, including the following:

- .NET Framework-reliant components
- ActiveX controls and plug-ins

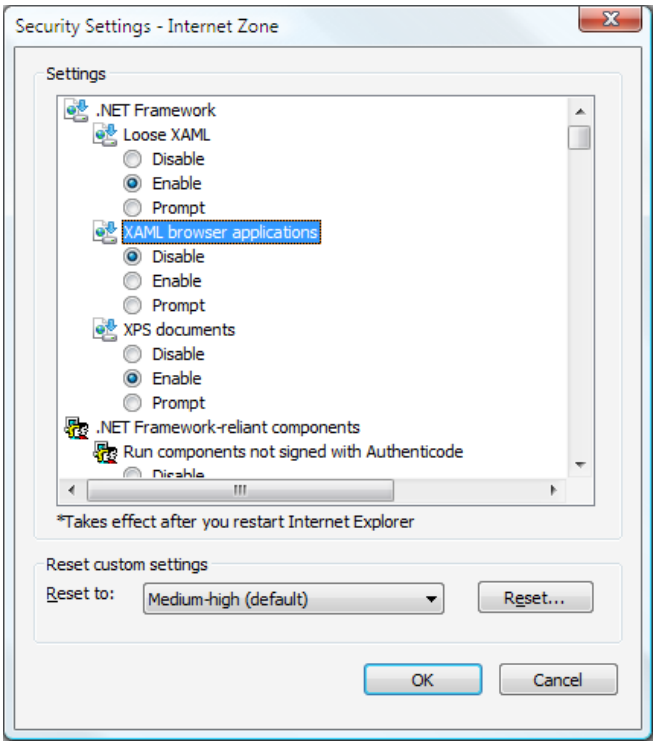
- Downloads
- Scripting
- User Authentication

The collection of functionality that can be secured in this way is configured on a per-zone basis for the **Internet**, **Intranet**, **Trusted Sites**, and **Restricted Sites** zones. The following steps describe how to configure your security settings:

1. Open **Control Panel**.
2. Click **Network and Internet** and then click **Internet Options**.

   The Internet Options dialog box appears.
3. On the **Security** tab, select the zone to configure the security settings for.
4. Click the **Custom Level** button.

   The **Security Settings** dialog box appears and you can configure the security settings for the selected zone.



> **NOTE**
> You can also get to the Internet Options dialog box from Internet Explorer. Click **Tools** and then click **Internet Options**.

Starting with Windows Internet Explorer 7, the following security settings specifically for .NET Framework are included:

- **Loose XAML**. Controls whether Internet Explorer can navigate to and loose XAML files. (Enable, Disable, and Prompt options).
- **XAML browser applications**. Controls whether Internet Explorer can navigate to and run XBAPs. (Enable, Disable, and Prompt options).

By default, these settings are all enabled for the **Internet**, **Local intranet**, and **Trusted sites** zones, and disabled for the **Restricted sites** zone.

Security-related WPF Registry Settings

In addition to the security settings available through the Internet Options, the following registry values are available for selectively blocking a number of security-sensitive WPF features. The values are defined under the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\Windows Presentation Foundation\Features
```

The following table lists the values that can be set.

VALUE NAME	VALUE TYPE	VALUE DATA
XBAPDisallow	REG_DWORD	1 to disallow; 0 to allow.
LooseXamlDisallow	REG_DWORD	1 to disallow; 0 to allow.
WebBrowserDisallow	REG_DWORD	1 to disallow; 0 to allow.
MediaAudioDisallow	REG_DWORD	1 to disallow; 0 to allow.
MediaImageDisallow	REG_DWORD	1 to disallow; 0 to allow.
MediaVideoDisallow	REG_DWORD	1 to disallow; 0 to allow.
ScriptInteropDisallow	REG_DWORD	1 to disallow; 0 to allow.

WebBrowser Control and Feature Controls

The WPF WebBrowser control can be used to host Web content. The WPF WebBrowser control wraps the underlying WebBrowser ActiveX control. WPF provides some support for securing your application when you use the WPF WebBrowser control to host untrusted Web content. However, some security features must be applied directly by the applications using the WebBrowser control. For more information about the WebBrowser ActiveX control, see WebBrowser Control Overviews and Tutorials.

> **NOTE**
> This section also applies to the Frame control since it uses the WebBrowser to navigate to HTML content.

If the WPF WebBrowser control is used to host untrusted Web content, your application should use a partial-trust AppDomain to help insulate your application code from potentially malicious HTML script code. This is especially true if your application is interacting with the hosted script by using the InvokeScript method and the ObjectForScripting property. For more information, see WPF Add-Ins Overview.

If your application uses the WPF WebBrowser control, another way to increase security and mitigate attacks is to enable Internet Explorer feature controls. Feature controls are additions to Internet Explorer that allow administrators and developers to configure features of Internet Explorer and applications that host the WebBrowser ActiveX control, which the WPF WebBrowser control wraps. Feature controls can be configured by using the CoInternetSetFeatureEnabled function or by changing values in the registry. For more information about feature controls, see Introduction to Feature Controls and Internet Feature Controls.

If you are developing a standalone WPF application that uses the WPF WebBrowser control, WPF automatically enables the following feature controls for your application.

FEATURE CONTROL
FEATURE_MIME_HANDLING
FEATURE_MIME_SNIFFING
FEATURE_OBJECT_CACHING
FEATURE_SAFE_BINDTOOBJECT

FEATURE CONTROL
FEATURE_WINDOW_RESTRICTIONS
FEATURE_ZONE_ELEVATION
FEATURE_RESTRICT_FILEDOWNLOAD
FEATURE_RESTRICT_ACTIVEXINSTALL
FEATURE_ADDON_MANAGEMENT
FEATURE_HTTP_USERNAME_PASSWORD_DISABLE
FEATURE_SECURITYBAND
FEATURE_UNC_SAVEDFILECHECK
FEATURE_VALIDATE_NAVIGATE_URL
FEATURE_DISABLE_TELNET_PROTOCOL
FEATURE_WEBOC_POPUPMANAGEMENT
FEATURE_DISABLE_LEGACY_COMPRESSION
FEATURE_SSLUX

Since these feature controls are enabled unconditionally, a full-trust application might be impaired by them. In this case, if there is no security risk for the specific application and the content it is hosting, the corresponding feature control can be disabled.

Feature controls are applied by the process instantiating the WebBrowser ActiveX object. Therefore, if you are creating a stand-alone application that can navigate to untrusted content, you should seriously consider enabling additional feature controls.

> **NOTE**
> This recommendation is based on general recommendations for MSHTML and SHDOCVW host security. For more information, see The MSHTML Host Security FAQ: Part I of II and The MSHTML Host Security FAQ: Part II of II.

For your executable, consider enabling the following feature controls by setting the registry value to 1.

FEATURE CONTROL
FEATURE_ACTIVEX_REPURPOSEDETECTION
FEATURE_BLOCK_LMZ_IMG
FEATURE_BLOCK_LMZ_OBJECT
FEATURE_BLOCK_LMZ_SCRIPT
FEATURE_RESTRICT_RES_TO_LMZ

FEATURE CONTROL
FEATURE_RESTRICT_ABOUT_PROTOCOL_IE7
FEATURE_SHOW_APP_PROTOCOL_WARN_DIALOG
FEATURE_LOCALMACHINE_LOCKDOWN
FEATURE_FORCE_ADDR_AND_STATUS
FEATURE_RESTRICTED_ZONE_WHEN_FILE_NOT_FOUND

For your executable, consider disabling the following feature control by setting the registry value to 0.

FEATURE CONTROL
FEATURE_ENABLE_SCRIPT_PASTE_URLACTION_IF_PROMPT

If you run a partial-trust XAML browser application (XBAP) that includes a WPF WebBrowser control in Windows Internet Explorer, WPF hosts the WebBrowser ActiveX control in the address space of the Internet Explorer process. Since the WebBrowser ActiveX control is hosted in the Internet Explorer process, all of the feature controls for Internet Explorer are also enabled for the WebBrowser ActiveX control.

XBAPs running in Internet Explorer also get an additional level of security compared to normal standalone applications. This additional security is because Internet Explorer, and therefore the WebBrowser ActiveX control, runs in protected mode by default on Windows Vista and Windows 7. For more information about protected mode, see Understanding and Working in Protected Mode Internet Explorer.

> **NOTE**
> If you try to run an XBAP that includes a WPF WebBrowser control in Firefox, while in the Internet zone, a SecurityException will be thrown. This is due to WPF security policy.

Disabling APTCA Assemblies for Partially Trusted Client Applications

When managed assemblies are installed into the global assembly cache (GAC), they become fully trusted because the user must provide explicit permission to install them. Because they are fully trusted, only fully trusted managed client applications can use them. To allow partially trusted applications to use them, they must be marked with the AllowPartiallyTrustedCallersAttribute (APTCA). Only assemblies that have been tested to be safe for execution in partial trust should be marked with this attribute.

However, it is possible for an APTCA assembly to exhibit a security flaw after being installed into the GAC . Once a security flaw is discovered, assembly publishers can produce a security update to fix the problem on existing installations, and to protect against installations that may occur after the problem is discovered. One option for the update is to uninstall the assembly, although that may break other fully trusted client applications that use the assembly.

WPF provides a mechanism by which an APTCA assembly can be disabled for partially trusted XBAPs without uninstalling the APTCA assembly.

To disable an APTCA assembly, you have to create a special registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\policy\APTCA\<AssemblyFullName>, FileVersion=
<AssemblyFileVersion>
```

The following shows an example:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\policy\APTCA\aptcagac, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=215e3ac809a0fea7, FileVersion=1.0.0.0
```

This key establishes an entry for the APTCA assembly. You also have to create a value in this key that enables or disables the assembly. The following are the details of the value:

- Value Name: **APTCA_FLAG**.
- Value Type: **REG_DWORD**.
- Value Data: **1** to disable; **0** to enable.

If an assembly has to be disabled for partially trusted client applications, you can write an update that creates the registry key and value.

> **NOTE**
> Core .NET Framework assemblies are not affected by disabling them in this way because they are required for managed applications to run. Support for disabling APTCA assemblies is primarily targeted to third-party applications.

Sandbox Behavior for Loose XAML Files

Loose XAML files are markup-only XAML files that do not depend on any code-behind, event handler, or application-specific assembly. When loose XAML files are navigated to directly from the browser, they are loaded in a security sandbox based on the default Internet zone permission set.

However, the security behavior is different when loose XAML files are navigated to from either a NavigationWindow or Frame in a standalone application.

In both cases, the loose XAML file that is navigated to inherits the permissions of its host application. However, this behavior may be undesirable from a security perspective, particularly if a loose XAML file was produced by an entity that is either not trusted or unknown. This type of content is known as *external content*, and both Frame and NavigationWindow can be configured to isolate it when navigated to. Isolation is achieved by setting the **SandboxExternalContent** property to true, as shown in the following examples for Frame and NavigationWindow:

```
<Frame
  Source="ExternalContentPage.xaml"
  SandboxExternalContent="True">
</Frame>
```

```
<!-- Sandboxing external content using NavigationWindow-->
<NavigationWindow
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Source="ExternalContentPage.xaml"
  SandboxExternalContent="True">
</NavigationWindow>
```

With this setting, external content will be loaded into a process that is separate from the process that is hosting the application. This process is restricted to the default Internet zone permission set, effectively isolating it from the hosting application and the client computer.

> **NOTE**
> Even though navigation to loose XAML files from either a NavigationWindow or Frame in a standalone application is implemented based on the WPF browser hosting infrastructure, involving the PresentationHost process, the security level is slightly less than when the content is loaded directly in Internet Explorer on Windows Vista and Windows 7 (which would still be through PresentationHost). This is because a standalone WPF application using a Web browser does not provide the additional Protected Mode security feature of Internet Explorer.

Resources for Developing WPF Applications that Promote Security

The following are some additional resources to help develop WPF applications that promote security:

AREA	RESOURCE
Managed code	Patterns and Practices Security Guidance for Applications
CAS	Code Access Security
ClickOnce	ClickOnce Security and Deployment
WPF	WPF Partial Trust Security

See also

- WPF Partial Trust Security
- WPF Security Strategy - Platform Security
- WPF Security Strategy - Security Engineering
- Patterns and Practices Security Guidance for Applications
- Code Access Security
- ClickOnce Security and Deployment
- XAML Overview (WPF)

WPF Partial Trust Security

11/3/2019 • 9 minutes to read • Edit Online

In general, Internet applications should be restricted from having direct access to critical system resources, to prevent malicious damage. By default, HTML and client-side scripting languages are not able to access critical system resources. Because Windows Presentation Foundation (WPF) browser-hosted applications can be launched from the browser, they should conform to a similar set of restrictions. To enforce these restrictions, WPF relies on both Code Access Security (CAS) and ClickOnce (see WPF Security Strategy - Platform Security). By default, browser-hosted applications request the Internet zone CAS set of permissions, irrespective of whether they are launched from the Internet, the local intranet, or the local computer. Applications that run with anything less than the full set of permissions are said to be running with partial trust.

WPF provides a wide variety of support to ensure that as much functionality as possible can be used safely in partial trust, and along with CAS, provides additional support for partial trust programming.

This topic contains the following sections:

WPF Feature Partial Trust Support

The following table lists the high-level features of Windows Presentation Foundation (WPF) that are safe to use within the limits of the Internet zone permission set.

Table 1: WPF Features that are Safe in Partial Trust

FEATURE AREA	FEATURE
General	Browser Window Site of Origin Access IsolatedStorage (512KB Limit) UIAutomation Providers Commanding Input Method Editors (IMEs) Tablet Stylus and Ink Simulated Drag/Drop using Mouse Capture and Move Events OpenFileDialog XAML Deserialization (via XamlReader.Load)

FEATURE AREA	FEATURE
Web Integration	Browser Download Dialog Top-Level User-Initiated Navigation mailto:links Uniform Resource Identifier Parameters HTTPWebRequest WPF Content Hosted in an IFRAME Hosting of Same-Site HTML Pages using Frame Hosting of Same Site HTML Pages using WebBrowser Web Services (ASMX) Web Services (using Windows Communication Foundation) Scripting Document Object Model
Visuals	2D and 3D Animation Media (Site Of Origin and Cross-Domain) Imaging/Audio/Video
Reading	FlowDocuments XPS Documents Embedded & System Fonts CFF & TrueType Fonts
Editing	Spell Checking RichTextBox Plaintext and Ink Clipboard Support User-Initiated Paste Copying Selected Content
Controls	General Controls

This table covers the WPF features at a high level. For more detailed information, the Windows SDK documents the permissions that are required by each member in WPF. Additionally, the following features have more detailed information regarding partial trust execution, including special considerations.

- XAML (see XAML Overview (WPF)).
- Popups (see System.Windows.Controls.Primitives.Popup).

- Drag and Drop (see Drag and Drop Overview).
- Clipboard (see System.Windows.Clipboard).
- Imaging (see System.Windows.Controls.Image).
- Serialization (see XamlReader.Load, XamlWriter.Save).
- Open File Dialog Box (see Microsoft.Win32.OpenFileDialog).

The following table outlines the WPF features that are not safe to run within the limits of the Internet zone permission set.

Table 2: WPF Features that are Not Safe in Partial Trust

FEATURE AREA	FEATURE
General	Window (Application Defined Windows and Dialog Boxes) SaveFileDialog File System Registry Access Drag and Drop XAML Serialization (via XamlWriter.Save) UIAutomation Clients Source Window Access (HwndHost) Full Speech Support Windows Forms Interoperability
Visuals	Bitmap Effects Image Encoding
Editing	Rich Text Format Clipboard Full XAML support

Partial Trust Programming

For XBAP applications, code that exceeds the default permission set will have different behavior depending on the security zone. In some cases, the user will receive a warning when they attempt to install it. The user can choose to continue or cancel the installation. The following table describes the behavior of the application for each security zone and what you have to do for the application to receive full trust.

SECURITY ZONE	BEHAVIOR	GETTING FULL TRUST
Local computer	Automatic full trust	No action is needed.
Intranet and trusted sites	Prompt for full trust	Sign the XBAP with a certificate so that the user sees the source in the prompt.

SECURITY ZONE	BEHAVIOR	GETTING FULL TRUST
Internet	Fails with "Trust Not Granted"	Sign the XBAP with a certificate.

> **NOTE**
> The behavior described in the previous table is for full trust XBAPs that do not follow the ClickOnce Trusted Deployment model.

In general, code that may exceed the allowed permissions is likely to be common code that is shared between both standalone and browser-hosted applications. CAS and WPF offer several techniques for managing this scenario.

Detecting Permissions Using CAS

In some situations, it is possible for shared code in library assemblies to be used by both standalone applications and XBAPs. In these cases, code may execute functionality that could require more permissions than the application's awarded permission set allows. Your application can detect whether or not it has a certain permission by using Microsoft .NET Framework security. Specifically, it can test whether it has a specific permission by calling the Demand method on the instance of the desired permission. This is shown in the following example, which has code that queries for whether it has the ability to save a file to the local disk:

```
using System.IO;
using System.IO.IsolatedStorage;
using System.Security;
using System.Security.Permissions;
using System.Windows;

namespace SDKSample
{
    public class FileHandling
    {
        public void Save()
        {
            if (IsPermissionGranted(new FileIOPermission(FileIOPermissionAccess.Write, @"c:\newfile.txt")))
            {
                // Write to local disk
                using (FileStream stream = File.Create(@"c:\newfile.txt"))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to local disk.");
                }
            }
            else
            {
                MessageBox.Show("I can't write to local disk.");
            }
        }

        // Detect whether or not this application has the requested permission
        bool IsPermissionGranted(CodeAccessPermission requestedPermission)
        {
            try
            {
                // Try and get this permission
                requestedPermission.Demand();
                return true;
            }
            catch
            {
                return false;
            }
        }
```

```
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows

Namespace SDKSample
    Public Class FileHandling
        Public Sub Save()
            If IsPermissionGranted(New FileIOPermission(FileIOPermissionAccess.Write, "c:\newfile.txt")) Then
                ' Write to local disk
                Using stream As FileStream = File.Create("c:\newfile.txt")
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to local disk.")
                End Using
                End Using
            Else
                MessageBox.Show("I can't write to local disk.")
            End If
        End Sub

        ' Detect whether or not this application has the requested permission
        Private Function IsPermissionGranted(ByVal requestedPermission As CodeAccessPermission) As Boolean
            Try
                ' Try and get this permission
                requestedPermission.Demand()
                Return True
            Catch
                Return False
            End Try
        End Function
```

```
    }
}
```

```
    End Class
End Namespace
```

If an application does not have the desired permission, the call to Demand will throw a security exception. Otherwise, the permission has been granted. `IsPermissionGranted` encapsulates this behavior and returns `true` or `false` as appropriate.

Graceful Degradation of Functionality

Being able to detect whether code has the permission to do what it needs to do is interesting for code that can be executed from different zones. While detecting the zone is one thing, it is far better to provide an alternative for the user, if possible. For example, a full trust application typically enables users to create files anywhere they want, while a partial trust application can only create files in isolated storage. If the code to create a file exists in an assembly that is shared by both full trust (standalone) applications and partial trust (browser-hosted) applications, and both applications want users to be able to create files, the shared code should detect whether it is running in partial or full trust before creating a file in the appropriate location. The following code demonstrates both.

```
using System.IO;
using System.IO.IsolatedStorage;
using System.Security;
using System.Security.Permissions;
using System.Windows;

namespace SDKSample
{
    public class FileHandlingGraceful
    {
        public void Save()
        {
            if (IsPermissionGranted(new FileIOPermission(FileIOPermissionAccess.Write, @"c:\newfile.txt")))
            {
                // Write to local disk
                using (FileStream stream = File.Create(@"c:\newfile.txt"))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to local disk.");
                }
            }
            else
            {
                // Persist application-scope property to
                // isolated storage
                IsolatedStorageFile storage = IsolatedStorageFile.GetUserStoreForApplication();
                using (IsolatedStorageFileStream stream =
                    new IsolatedStorageFileStream("newfile.txt", FileMode.Create, storage))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to Isolated Storage");
                }
            }
        }

        // Detect whether or not this application has the requested permission
        bool IsPermissionGranted(CodeAccessPermission requestedPermission)
        {
            try
            {
                // Try and get this permission
                requestedPermission.Demand();
                return true;
            }
            catch
            {
                return false;
            }
        }
```

```
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows

Namespace SDKSample
    Public Class FileHandlingGraceful
        Public Sub Save()
            If IsPermissionGranted(New FileIOPermission(FileIOPermissionAccess.Write, "c:\newfile.txt")) Then
                ' Write to local disk
                Using stream As FileStream = File.Create("c:\newfile.txt")
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to local disk.")
                End Using
                End Using
            Else
                ' Persist application-scope property to
                ' isolated storage
                Dim storage As IsolatedStorageFile = IsolatedStorageFile.GetUserStoreForApplication()
                Using stream As New IsolatedStorageFileStream("newfile.txt", FileMode.Create, storage)
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to Isolated Storage")
                End Using
                End Using
            End If
        End Sub

        ' Detect whether or not this application has the requested permission
        Private Function IsPermissionGranted(ByVal requestedPermission As CodeAccessPermission) As Boolean
            Try
                ' Try and get this permission
                requestedPermission.Demand()
                Return True
            Catch
                Return False
            End Try
        End Function
```

```
    }
}
```

```
    End Class
End Namespace
```

In many cases, you should be able to find a partial trust alternative.

In a controlled environment, such as an intranet, custom managed frameworks can be installed across the client base into the global assembly cache (GAC). These libraries can execute code that requires full trust, and be referenced from applications that are only allowed partial trust by using AllowPartiallyTrustedCallersAttribute (for more information, see Security and WPF Security Strategy - Platform Security).

Browser Host Detection

Using CAS to check for permissions is a suitable technique when you need to check on a per-permission basis. Although, this technique depends on catching exceptions as a part of normal processing, which is not recommended in general and can have performance issues. Instead, if your XAML browser application (XBAP) only runs within the Internet zone sandbox, you can use the BrowserInteropHelper.IsBrowserHosted property, which returns true for XAML browser applications (XBAPs).

NOTE

IsBrowserHosted only distinguishes whether an application is running in a browser, not which set of permissions an application is running with.

Managing Permissions

By default, XBAPs run with partial trust (default Internet zone permission set). However, depending on the requirements of the application, it is possible to change the set of permissions from the default. For example, if an XBAPs is launched from a local intranet, it can take advantage of an increased permission set, which is shown in the following table.

Table 3: LocalIntranet and Internet Permissions

PERMISSION	ATTRIBUTE	LOCALINTRANET	INTERNET
DNS	Access DNS servers	Yes	No
Environment Variables	Read	Yes	No
File Dialogs	Open	Yes	Yes
File Dialogs	Unrestricted	Yes	No
Isolated Storage	Assembly isolation by user	Yes	No
Isolated Storage	Unknown isolation	Yes	Yes
Isolated Storage	Unlimited user quota	Yes	No
Media	Safe audio, video, and images	Yes	Yes
Printing	Default printing	Yes	No
Printing	Safe printing	Yes	Yes
Reflection	Emit	Yes	No
Security	Managed code execution	Yes	Yes
Security	Assert granted permissions	Yes	No
User Interface	Unrestricted	Yes	No
User Interface	Safe top level windows	Yes	Yes
User Interface	Own Clipboard	Yes	Yes
Web Browser	Safe frame navigation to HTML	Yes	Yes

> **NOTE**
>
> Cut and Paste is only allowed in partial trust when user initiated.

If you need to increase permissions, you need to change the project settings and the ClickOnce application manifest. For more information, see WPF XAML Browser Applications Overview. The following documents may also be helpful.

- Mage.exe (Manifest Generation and Editing Tool).
- MageUI.exe (Manifest Generation and Editing Tool, Graphical Client).
- Securing ClickOnce Applications.

If your XBAP requires full trust, you can use the same tools to increase the requested permissions. Although an XBAP will only receive full trust if it is installed on and launched from the local computer, the intranet, or from a URL that is listed in the browser's trusted or allowed sites. If the application is installed from the intranet or a trusted site, the user will receive the standard ClickOnce prompt notifying them of the elevated permissions. The user can choose to continue or cancel the installation.

Alternatively, you can use the ClickOnce Trusted Deployment model for full trust deployment from any security zone. For more information, see Trusted Application Deployment Overview and Security.

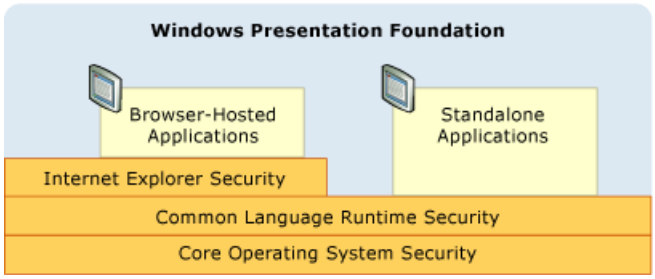
See also

- Security
- WPF Security Strategy - Platform Security
- WPF Security Strategy - Security Engineering

WPF Security Strategy - Platform Security

11/1/2019 • 13 minutes to read • Edit Online

While Windows Presentation Foundation (WPF) provides a variety of security services, it also leverages the security features of the underlying platform, which includes the operating system, the CLR, and Internet Explorer. These layers combine to provide WPF a strong, defense-in-depth security model that attempts to avoid any single point of failure, as shown in the following figure:



The remainder of this topic discusses the features in each of these layers that pertain to WPF specifically.

Operating System Security

The core of Windows provides several security features that form the security foundation for all Windows applications, including those built with WPF. This topic discusses the breadth of these security features that are important to WPF, as well as how WPF integrates with them to provide further defense-in-depth.

Microsoft Windows XP Service Pack 2 (SP2)

In addition to a general review and strengthening of Windows, there are three key features from Windows XP SP2 that we will discuss in this topic:

- /GS compilation
- Microsoft Windows Update.

/GS Compilation

Windows XP SP2 provides protection by recompiling many core system libraries, including all of the WPF dependencies such as the CLR, to help mitigate buffer overruns. This is achieved by using the /GS parameter with the C/C++ command-line compiler. Although buffer overruns should be explicitly avoided, /GS compilation provides an example of a defense-in-depth against potential vulnerabilities that are inadvertently or maliciously created by them.

Historically, buffer overruns have been the cause of many high-impact security exploits. A buffer overrun occurs when an attacker takes advantage of a code vulnerability that allows the injection of malicious code that writes past the boundaries of a buffer. This then allows an attacker to hijack the process in which the code is executing by overwriting the return address of a function to cause the execution of the attacker's code. The result is malicious code that executes arbitrary code with the same privileges as the hijacked process.

At a high level, the -GS compiler flag protects against some potential buffer overruns by injecting a special security cookie to protect the return address of a function that has local string buffers. After a function returns, the security cookie is compared with its previous value. If the value has changed, a buffer overrun may have occurred and the process is stopped with an error condition. Stopping the process prevents the execution of potentially malicious code. See -GS (Buffer Security Check) for more details.

WPF is compiled with the /GS flag to add yet another layer of defense to WPF applications.

Windows Vista

WPF users on Windows Vista will benefit from the operating system's additional security enhancements, including "Least-Privilege User Access", code integrity checks, and privilege isolation.

User Account Control (UAC)

Today, Windows users tend to run with administrator privileges because many applications require them for either installation or execution, or both. Being able to write default application settings to the Registry is one example.

Running with administrator privileges really means that applications execute from processes that are granted administrator privileges. The security impact of this is that any malicious code that hijacks a process running with administrator privileges will automatically inherit those privileges, including access to critical system resources.

One way to protect against this security threat is to run applications with the least amount of privileges that are required. This is known as the principle of least privilege, and is a core feature of the Windows operating system. This feature is called User Account Control (UAC), and is used by Windows UAC in two key ways:

- To run most applications with UAC privileges by default, even if the user is an administrator; only applications that need administrator privileges will run with administrator privileges. To run with administrative privileges, applications must be explicitly marked in either their application manifest or as an entry in security policy.
- To provide compatibility solutions like virtualization. For example, many applications try to write to restricted locations like C:\Program Files. For applications executing under UAC, an alternative per-user location exists that does not require administrator privileges to write to. For applications running under UAC, UAC virtualizes C:\Program Files so that applications who think they are writing to it are actually writing to the alternative, per-user location. This kind of compatibility work enables the operating system to run many applications that couldn't previously run in UAC.

Code Integrity Checks

Windows Vista incorporates deeper code integrity checks to help prevent malicious code from being injected into system files or into the kernel at load/run time. This goes beyond system file protection.

Limited Rights Process for Browser-Hosted Applications

Browser-hosted WPF applications execute within the Internet zone sandbox. WPF integration with Microsoft Internet Explorer extends this protection with additional support.

Since XAML browser applications (XBAPs) are generally sandboxed by the Internet zone permission set, removing these privileges does not harm XAML browser applications (XBAPs) from a compatibility perspective. Instead, an additional defense-in-depth layer is created; if a sandboxed application is able to exploit other layers and hijack the process, the process will still only have limited privileges.

See Using a Least-Privileged User Account.

Common Language Runtime Security

The common language runtime (CLR) offers a number of key security benefits that include validation and verification, Code Access Security (CAS), and the Security Critical Methodology.

Validation and Verification

To provide assembly isolation and integrity, the CLR uses a process of validation. CLR validation ensures that assemblies are isolated by validating their Portable Executable (PE) file format for addresses that point outside the assembly. CLR validation also validates the integrity of the metadata that is embedded within an assembly.

To ensure type safety, help prevent common security issues (e.g. buffer overruns), and enable sandboxing through sub-process isolation, CLR security uses the concept of verification.

Managed applications are compiled into Microsoft Intermediate Language (MSIL). When methods in a managed application are executed, its MSIL is compiled into native code through Just-In-Time (JIT) compilation. JIT compilation includes a verification process that applies many safety and robustness rules that ensure code does not:

- Violate type contracts
- Introduce buffer overruns
- Wildly access memory.

Managed code that does not conform to verification rules is not allowed to execute, unless it is considered trusted code.

The advantage of verifiable code is a key reason why WPF builds on the .NET Framework. To the extent that verifiable code is used, the possibility of exploiting possible vulnerabilities is greatly lowered.

Code Access Security

A client machine exposes a wide variety of resources that a managed application can have access to, including the file system, the Registry, printing services, the user interface, reflection, and environment variables. Before a managed application can access any of the resources on a client machine, it must have .NET Framework permission to do so. A permission in CAS is a subclass of the CodeAccessPermission; CAS implements one subclass for each resource that managed applications can access.

The set of permissions that a managed application is granted by CAS when it starts executing is known as a permission set and is determined by evidence provided by the application. For WPF applications, the evidence that is provided is the location, or zone, from which the applications are launched. CAS identifies the following zones:

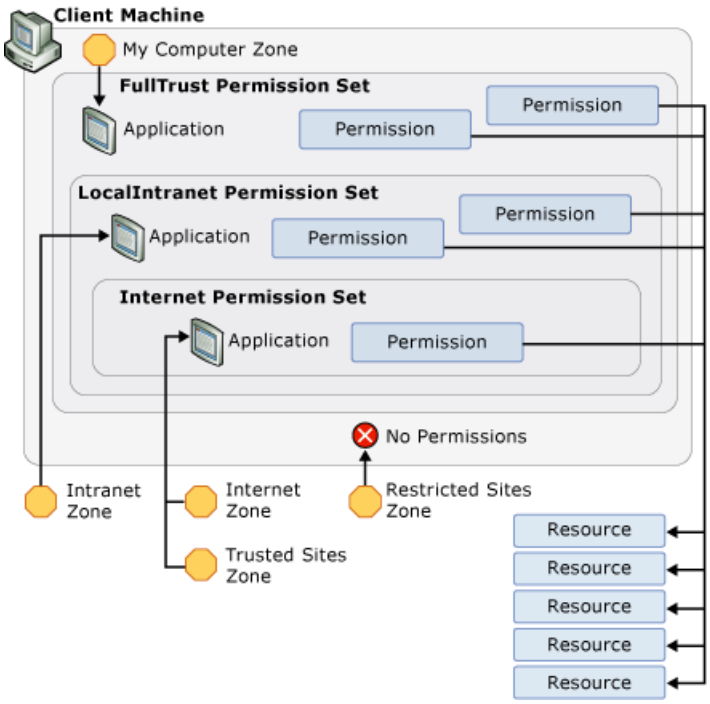
- **My Computer**. Applications launched from the client machine (Fully Trusted).
- **Local Intranet**. Applications launched from the intranet. (Somewhat Trusted).
- **Internet**. Applications launched from the Internet. (Least Trusted).
- **Trusted Sites**. Applications identified by a user as being trusted. (Least Trusted).
- **Untrusted Sites**. Applications identified by a user as being untrusted. (Untrusted).

For each of these zones, CAS provides a predefined permission set that includes the permissions which matches the level of trust associated with each. These include:

- **FullTrust**. For applications launched from the **My Computer** zone. All possible permissions are granted.
- **LocalIntranet**. For applications launched from the **Local Intranet** zone. A subset of permissions are granted to provide moderate access to a client machine's resources, including isolated storage, unrestricted UI access, unrestricted file dialogs, limited reflection, limited access to environment variables. Permissions for critical resources like the Registry are not provided.
- **Internet**. For applications launched from the **Internet** or **Trusted Sites** zone. A subset of permissions are granted to provided limited access to a client machine's resources, including isolated storage, file open only, and limited UI. Essentially, this permission set isolates applications from the client machine.

Applications identified as being from the **Untrusted Sites** zone are granted no permissions by CAS at all. Consequently, a predefined permission set does not exist for them.

The following figure illustrates the relationship between zones, permission sets, permissions, and resources:



The restrictions of the Internet zone security sandbox apply equally to any code that an XBAP imports from a system library, including WPF. This ensures that every bit of the code is locked down, even WPF. Unfortunately, to be able to execute, an XBAP needs to execute functionality that requires more permissions than those enabled by the Internet zone security sandbox.

Consider an XBAP application that includes the following page:

```
FileIOPermission fp = new FileIOPermission(PermissionState.Unrestricted);
fp.Assert();

// Perform operation that uses the assert

// Revert the assert when operation is completed
CodeAccessPermission.RevertAssert();
```

```
Dim fp As New FileIOPermission(PermissionState.Unrestricted)
fp.Assert()

' Perform operation that uses the assert

' Revert the assert when operation is completed
CodeAccessPermission.RevertAssert()
```

To execute this XBAP, the underlying WPF code must execute more functionality than is available to the calling XBAP, including:

- Creating a window handle (HWND) for rendering
- Dispatching messages
- Loading the Tahoma font

From a security point of view, allowing direct access to any of these operations from the sandboxed application would be catastrophic.

Fortunately, WPF caters to this situation by allowing these operations to execute with elevated privileges on behalf of the sandboxed application. While all WPF operations are checked against the limited Internet zone security permissions of the application domain of the XBAP, WPF (as with other system libraries) is granted a

permission set that includes all possible permissions.

This requires that WPF receives elevated privileges while preventing those privileges from being governed by the Internet zone permission set of the host application domain.

WPF does this by using the **Assert** method of a permission. The following code shows how this happens.

```
FileIOPermission fp = new FileIOPermission(PermissionState.Unrestricted);
fp.Assert();

// Perform operation that uses the assert

// Revert the assert when operation is completed
CodeAccessPermission.RevertAssert();
```

```
Dim fp As New FileIOPermission(PermissionState.Unrestricted)
fp.Assert()

' Perform operation that uses the assert

' Revert the assert when operation is completed
CodeAccessPermission.RevertAssert()
```

The **Assert** essentially prevents the unlimited permissions required by WPF from being restricted by the Internet zone permissions of the XBAP.

From a platform perspective, WPF is responsible for using **Assert** correctly; an incorrect use of **Assert** could enable malicious code to elevate privileges. Consequently, it is important then to only call **Assert** when needed, and to ensure that sandbox restrictions remain intact. For example, sandboxed code is not allowed to open random files, but it is allowed to use fonts. WPF enables sandboxed applications to use font functionality by calling **Assert**, and for WPF to read files known to contain those fonts on behalf of the sandboxed application.

ClickOnce Deployment

ClickOnce is a comprehensive deployment technology that is included with .NET Framework, and integrates with Visual Studio (see ClickOnce security and deployment for detailed information). Standalone WPF applications can be deployed using ClickOnce, while browser-hosted applications must be deployed with ClickOnce.

Applications deployed using ClickOnce are given an additional security layer over Code Access Security (CAS); essentially, ClickOnce deployed applications request the permissions that they need. They are granted only those permissions if they do not exceed the set of permissions for the zone from which the application is deployed. By reducing the set of permissions to only those that are needed, even if they are less than those provided by the launch zone's permission set, the number of resources that the application has access to is reduced to a bare minimum. Consequently, if the application is hijacked, the potential for damage to the client machine is reduced.

Security-Critical Methodology

The WPF code that uses permissions to enable the Internet zone sandbox for XBAP applications must be held to highest possible degree of security audit and control. To facilitate this requirement, .NET Framework provides new support for managing code that elevates privilege. Specifically, the CLR enables you to identify code that elevates privilege and mark it with the SecurityCriticalAttribute; any code not marked with SecurityCriticalAttribute becomes *transparent* using this methodology. Conversely, managed code that is not marked with SecurityCriticalAttribute is prevented from elevating privilege.

The Security-Critical Methodology allows the organization of WPF code that elevates privilege into *security-critical kernel*, with the remainder being transparent. Isolating the security-critical code enables the WPF engineering team focus an additional security analysis and source control on the security-critical kernel above and beyond standard security practices (see WPF Security Strategy - Security Engineering).

Note that .NET Framework permits trusted code to extend the XBAP Internet zone sandbox by allowing developers to write managed assemblies that are marked with AllowPartiallyTrustedCallersAttribute (APTCA) and deployed to the user's Global Assembly Cache (GAC). Marking an assembly with APTCA is a highly sensitive security operation as it allows any code to call that assembly, including malicious code from the Internet. Extreme caution and best practices must be used when doing this and users must choose to trust that software in order for it to be installed.

Microsoft Internet Explorer Security

Beyond reducing security issues and simplifying security configuration, Microsoft Internet Explorer 6 (SP2) contains several features that security improvements that enhance security for users of XAML browser applications (XBAPs). The thrust of these features attempts to allow users greater control over their browsing experience.

Prior to IE6 SP2, users could be subject to any of the following:

- Random popup windows.
- Confusing script redirection.
- Numerous security dialogs on some Web sites.

In some cases, untrustworthy Web sites would try to trick users by spoofing installation user interface (UI) or repeatedly showing a Microsoft ActiveX installation dialog box, even though the user may have canceled it. Using these techniques, it is possible that a significant number of users have been tricked into making poor decisions that resulted with the installation of spyware applications.

IE6 SP2 includes several features to mitigate these types of issues, which revolve around the concept of user initiation. IE6 SP2 detects when a user has clicked on a link or page element prior to an action, which is known as *user initiation*, and treats it differently than when a similar action is instead triggered by the script on a page. As an example, IE6 SP2 incorporates a **Pop-Up Blocker** that detects when a user clicks a button prior to the page creating a pop-up. This enables IE6 SP2 to allow most innocuous pop-ups while preventing pop-ups that users neither ask for nor want. Blocked pop-ups are trapped under the new **Information Bar**, which allows the user to manually override the block and view the pop-up.

The same user-initiation logic is also applied to **Open**/**Save** security prompts. ActiveX installation dialog boxes are always trapped under the Information Bar unless they represent an upgrade from a previously installed control. These measures combine to give users a safer, more controlled user experience since they are protected against sites which harass them to install either unwanted or malicious software.

These features also protect customers who use IE6 SP2 to browse to web sites that allow them to download and install WPF applications. In particular, this is because IE6 SP2 offers a better user experience that reduces the chance for users to install malicious or devious applications irrespective of what technology was used to build it, including WPF. WPF adds to these protections by using ClickOnce to facilitate downloading of its applications over the Internet. Since XAML browser applications (XBAPs) execute within an Internet zone security sandbox, they can be seamlessly launched. On the other hand, standalone WPF applications require full trust to execute. For these applications, ClickOnce will display a security dialog box during the launch process to notify the use of the application's additional security requirements. However, this must be user-initiated, will also be governed by user initiated logic, and can be canceled.

Internet Explorer 7 incorporates and extends the security capabilities of IE6 SP2 as part of an ongoing commitment to security.

See also

- Code Access Security

- Security
- WPF Partial Trust Security
- WPF Security Strategy - Security Engineering

WPF Security Strategy - Security Engineering

11/1/2019 • 3 minutes to read • Edit Online

Trustworthy Computing is a Microsoft initiative for ensuring the production of secure code. A key element of the Trustworthy Computing initiative is the Microsoft Security Development Lifecycle (SDL). The SDL is an engineering practice that is used in conjunction with standard engineering processes to facilitate the delivery of secure code. The SDL consists of ten phases that combine best practices with formalization, measurability, and additional structure, including:

- Security design analysis
- Tool-based quality checks
- Penetration testing
- Final security review
- Post release product security management

WPF Specifics

The WPF engineering team both applies and extends the SDL, the combination of which includes the following key aspects:

Threat Modeling

Security Analysis and Editing Tools

Testing Techniques

Critical Code Management

Threat Modeling

Threat modeling is a core component of the SDL, and is used to profile a system to determine potential security vulnerabilities. Once the vulnerabilities are identified, threat modeling also ensures that appropriate mitigations are in place.

At a high level, threat modeling involves the following key steps by using a grocery store as an example:

1. **Identifying Assets**. A grocery store's assets might include employees, a safe, cash registers, and inventory.
2. **Enumerating Entry Points**. A grocery store's entry points might include the front and back doors, windows, the loading dock, and air conditioning units.
3. **Investigating Attacks against Assets using Entry Points**. One possible attack could target a grocery store's *safe* asset through the *air conditioning* entry point; the air conditioning unit could be unscrewed to allow the safe to be pulled up through it and out of the store.

Threat modeling is applied throughout WPF and includes the following:

- How the XAML parser reads files, maps text to corresponding object model classes, and creates the actual code.
- How a window handle (hWnd) is created, sends messages, and is used for rendering the contents of a window.

- How data binding obtains resources and interacts with the system.

These threat models are important for identifying security design requirements and threat mitigations during the development process.

Source Analysis and Editing Tools

In addition to the manual security code review elements of the SDL, the WPF team uses several tools for source analysis and associated edits to decrease security vulnerabilities. A wide range of source tools are used, and include the following:

- **FXCop**: Finds common security issues in managed code ranging from inheritance rules to code access security usage to how to safely interoperate with unmanaged code. See FXCop.
- **Prefix/Prefast**: Finds security vulnerabilities and common security issues in unmanaged code such as buffer overruns, format string issues, and error checking.
- **Banned APIs**: Searches source code to identify accidental usage of functions that are well-known for causing security issues, such as `strcpy`. Once identified, these functions are replaced with alternatives that are more secure.

Testing Techniques

WPF uses a variety of security testing techniques that include:

- **Whitebox Testing**: Testers view source code, and then build exploit tests.
- **Blackbox Testing**: Testers try to find security exploits by examining the API and features, and then try to attack the product.
- **Regressing Security Issues from other Products**: Where relevant, security issues from related products are tested. For example, appropriate variants of approximately sixty security issues for Internet Explorer have been identified and tried for their applicability to WPF.
- **Tools-Based Penetration Testing through File Fuzzing**: File fuzzing is the exploitation of a file reader's input range through a variety of inputs. One example in WPF where this technique is used is to check for failure in image decoding code.

Critical Code Management

For XAML browser applications (XBAPs), WPF builds a security sandbox by using .NET Framework support for marking and tracking security-critical code that elevates privileges (see **Security-Critical Methodology** in WPF Security Strategy - Platform Security). Given the high security quality requirements on security critical code, such code receives an additional level of source management control and security audit. Approximately 5% to 10% of WPF consists of security-critical code, which is reviewed by a dedicated reviewing team. The source code and check-in process is managed by tracking security critical code and mapping each critical entity (i.e. a method that contains critical code) to its sign off state. The sign off state includes the names of one or more reviewers. Each daily build of WPF compares the critical code to that in previous builds to check for unapproved changes. If an engineer modifies critical code without approval from the reviewing team, it is identified and fixed immediately. This process enables the application and maintenance of an especially high level of scrutiny over WPF sandbox code.

See also

- Security
- WPF Partial Trust Security
- WPF Security Strategy - Platform Security
- Trustworthy Computing
- Security in .NET

WPF Samples

11/10/2018 • 2 minutes to read • Edit Online

For samples that demonstrate Windows Presentation Foundation (WPF), see the Microsoft/WPF-Samples repo on GitHub.

Class Library (WPF)

7/12/2019 • 2 minutes to read • Edit Online

The following links refer to namespaces that contain Windows Presentation Foundation (WPF) APIs.

In This Section

Reference

- Microsoft.Build.Tasks.Windows
- Microsoft.Win32 (shared)
- Microsoft.Windows.Themes
- System.Collections.ObjectModel (shared)
- System.Collections.Specialized (shared)
- System.ComponentModel (shared)
- System.Diagnostics (shared)
- System.IO (shared)
- System.IO.Packaging
- System.Printing
- System.Printing.IndexedProperties
- System.Printing.Interop
- System.Security.Permissions (shared)
- System.Security.RightsManagement
- System.Windows
- System.Windows.Annotations
- System.Windows.Annotations.Storage
- System.Windows.Automation
- System.Windows.Automation.Peers
- System.Windows.Automation.Provider
- System.Windows.Automation.Text
- System.Windows.Controls
- System.Windows.Controls.Primitives
- System.Windows.Converters
- System.Windows.Data

- System.Windows.Documents
- System.Windows.Documents.DocumentStructures
- System.Windows.Documents.Serialization
- System.Windows.Forms.Integration
- System.Windows.Ink
- System.Windows.Input
- System.Windows.Input.StylusPlugIns
- System.Windows.Interop
- System.Windows.Markup (shared)
- System.Windows.Markup.Localizer
- System.Windows.Markup.Primitives
- System.Windows.Media
- System.Windows.Media.Animation
- System.Windows.Media.Converters
- System.Windows.Media.Effects
- System.Windows.Media.Imaging
- System.Windows.Media.Media3D
- System.Windows.Media.Media3D.Converters
- System.Windows.Media.TextFormatting
- System.Windows.Navigation
- System.Windows.Resources
- System.Windows.Shapes
- System.Windows.Threading
- System.Windows.Xps
- System.Windows.Xps.Packaging
- System.Windows.Xps.Serialization
- UIAutomationClientsideProviders

XAML Support in .NET 4

The following namespaces contain types from the System.Xaml assembly. System.Xaml provides common XAML language support for frameworks such as WPF that are built on .NET Framework 4.

- System.Windows.Markup (shared)
- System.Xaml
- System.Xaml.Permissions

- System.Xaml.Schema