

Contents

Design user interfaces

Design XAML in Visual Studio

XAML Designer

- Work with elements

- Organize objects into layout containers

- Create and apply a resource

- Walkthrough: Bind to data in XAML Designer

- Keyboard shortcuts for XAML Designer

- Debug or disable project code

- XAML errors and warnings

Blend for Visual Studio

- Insert controls and modify their behavior

- Insert images, videos, and audio clips

- Draw shapes and paths

- Modify the style of objects in Blend

- Animate objects

- Display data

- Keyboard shortcuts and modifier keys

 - Keyboard shortcuts

 - Artboard modifier keys

 - Pen tool modifier keys

 - Direct selection tool modifier keys

- Accessibility products and services for Blend

Windows Presentation Foundation (WPF)

WPF overview

WPF data binding with LINQ to XML

- LINQ to XML dynamic properties

 - XAttribute class dynamic properties

 - Value (XAttribute dynamic property)

XElement class dynamic properties

Attribute (XElement dynamic property)

Element (XElement dynamic property)

Elements (XElement dynamic property)

Descendants (XElement dynamic property)

Value (XElement dynamic property)

Xml (XElement dynamic property)

Example

Build and run the LinqToXmlDataBinding example

Walkthrough: LINQ to XML data binding

Work with 3D assets for games and apps

Work with textures and images

Image editor

Image editor examples

Create a basic texture

Create and modify MIP levels

Work with 3D models

Model editor

Model editor examples

Create a basic 3D model

Modify the pivot point of a 3D model

Model 3D terrain

Apply a shader to a 3D model

Work with shaders

Shader designer

Shader designer nodes

Constant nodes

Parameter nodes

Texture nodes

Math nodes

Utility nodes

Filter nodes

Shader designer examples

- Create a basic color shader

- Create a basic Lambert shader

- Create a basic Phong shader

- Create a basic texture shader

- Create a grayscale texture shader

- Create a geometry-based gradient shader

- Walkthrough: Create a realistic 3D billiard ball

- Export a shader

Use 3D assets in your game or app

- Export a texture that contains mipmaps

- Export a texture that has premultiplied alpha

- Export a texture for use with Direct2D or Javascript apps

The Visual Studio image library

Develop apps with Workflow Designer

Design user interfaces

2/8/2019 • 2 minutes to read • [Edit Online](#)

You can create and design the user interface for your application by using a variety of tools in Visual Studio.

TO LEARN MORE ABOUT	SEE
The features of the XAML designers in Visual Studio and Blend for Visual Studio	Design XAML in Visual Studio and Blend for Visual Studio
Designing any XAML-based app using Visual Studio	Create a UI by using XAML Designer in Visual Studio
Designing any XAML-based app using Blend for Visual Studio	Create a UI by using Blend for Visual Studio
Designing desktop applications that use the WPF flavor of XAML	Get started with Windows Presentation Foundation
Developing a DirectX application in Visual Studio	Work with 3D assets for games and apps
Standard icons available for your programs	The Visual Studio image library

Design XAML in Visual Studio

2/8/2019 • 5 minutes to read • [Edit Online](#)

Visual Studio and Blend for Visual Studio both provide visual tools for building engaging user interfaces and rich media experiences with XAML for a variety of app types. Both tools share a common set of features including a visual XAML editor, but Blend for Visual Studio provides additional design tools for more advanced tasks such as animation and behaviors.

The process of designing an app depends on the tool you choose and your target platform. This article compares the XAML design tools in Visual Studio and Blend for Visual Studio. For more detailed walkthroughs of using the tools, see the following topics:

- [Create a UI by using XAML Designer in Visual Studio](#)
- [Create a UI by using Blend for Visual Studio](#)

Choose the right tool

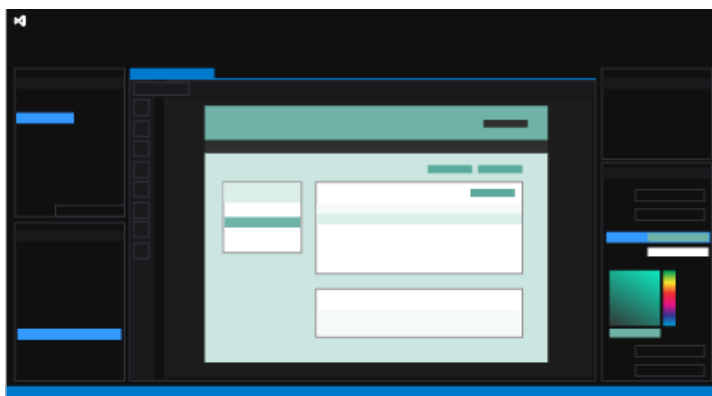
Your choice of design tools is largely dependent on your skill set. If you are more code-oriented, you can write XAML code in Visual Studio to accomplish advanced design tasks. If you are more design-oriented, Blend for Visual Studio lets you perform advanced tasks without writing code.

You can switch back and forth between Visual Studio and Blend for Visual Studio, and you can even have the same project open in both at the same time. Changes made to XAML files in one IDE can be applied via automatic reload when you switch to the other IDE. You can control the reload behavior via options in the **Tools > Options** dialog box in either IDE.

Shared Capabilities

For most basic tasks, the IDE for Visual Studio and Blend for Visual Studio share the same set of windows and capabilities, with some subtle differences. Some highlights include:

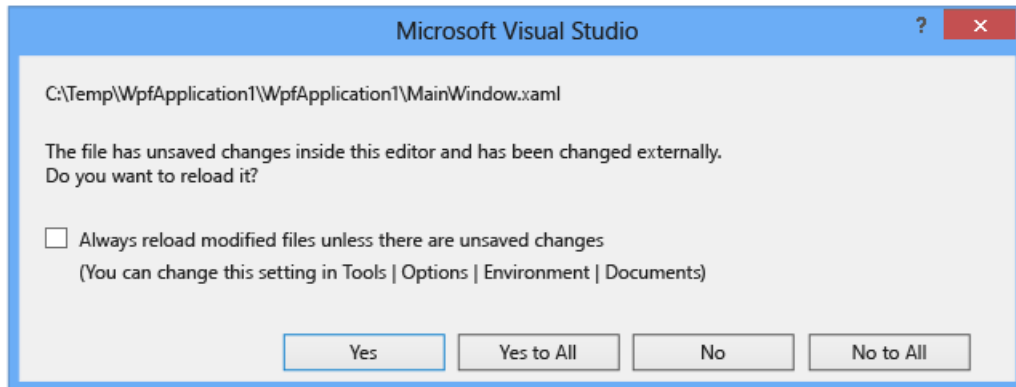
- **A consistent user interface:** You can design your applications within the familiar context of the Visual Studio user interface, which makes switching between IDEs a more pleasant and productive experience. Blend for Visual Studio uses the Visual Studio Dark theme that helps you focus on the content you are designing by improving the contrast between your content and the user interface. See [Create a UI by using XAML Designer](#).



- **XAML IntelliSense:** Both IDEs support all of the common capabilities you would expect from IntelliSense including statement completion, support for common editor operations like commenting and formatting code, and navigation to resources, binding, and code.
- **Basic debugging capabilities:** You can now debug in Blend, including setting breakpoints in your code to

debug your running app. To maintain a consistent debugging experience with Visual Studio, Blend for Visual Studio includes most of Visual Studio's debugging windows and toolbars. Advanced debugging capabilities such as diagnostics and code analysis are only available in Visual Studio. See [Debug in Visual Studio](#).

- **File reload experience:** You can edit your XAML files in either Blend for Visual Studio or Visual Studio, and have your edited files reload automatically as you switch between them. To minimize workflow interruptions, you can now set your file reload preferences in the file reload dialog.



- **Synchronized Layouts and Settings:** Custom layouts enable you to save and apply tool window layout customizations. Visual Studio synchronizes these customizations and preferences for both Visual Studio and Blend for Visual Studio across machines when you sign in with the same Microsoft account. See [Synchronize settings across multiple computers](#).
- **A common Solution Explorer: Solution Explorer** provides you with an organized view of your projects and their files, as well as ready access to the commands associated with them. With Solution Explorer, it is easier to work with big enterprise projects. See [Solutions and projects](#).
- **Team Explorer:** With Team Explorer you can manage your projects with GIT or TFS repositories to facilitate team collaboration. See [Work in Team Explorer](#).
- **NuGet:** You can manage NuGet packages in both Visual Studio and Blend for Visual Studio. NuGet is a package manager for the .NET Framework that simplifies the installation and removal of packages from a solution.

Advanced Capabilities in Blend for Visual Studio

To increase your productivity, consider using Blend for Visual Studio for the following tasks. These are the areas where Blend for Visual Studio offers more speed and functionality than the Visual Studio designer or code alone.

TO	VISUAL STUDIO	BLEND FOR VISUAL STUDIO	MORE INFORMATION
Create animations	There is no design tool for animations; you have to create them programmatically. This requires an understanding of the animation and timing system in WPF and extensive coding expertise.	You create animations visually and can preview them in Blend for Visual Studio. This is faster and more accurate than building your animations in code. You can add triggers to handle user interaction, and you can switch to code to add event handlers and other functionality.	Animate objects

TO	VISUAL STUDIO	BLEND FOR VISUAL STUDIO	MORE INFORMATION
Turn shapes and text into paths for easier manipulation	Not supported.	<p>You can make subtle or dramatic changes to shapes (such as rectangles and ellipses) by converting them to paths, which provide better editing control. You can reshape or combine paths, and create compound paths from multiple shapes.</p> <p>You can also convert text blocks into paths to manipulate them as vector images.</p>	Draw shapes and paths
Add interactivity to your UI designs	Requires C#, Visual Basic, or C++ code.	<p>Drag and drop behaviors onto controls to add interactivity to your static designs. Behaviors are ready-to-use code snippets that encapsulate functionality such as drag/drop, zoom, and visual state changes. There's a growing set of behaviors from which you can choose, and you can create your own.</p> <p>You can then customize each behavior by changing its properties in Blend for Visual Studio or by adding event handlers in code.</p>	Insert controls and modify their behavior
Use Adobe artwork	Not supported.	Import Adobe FXG, PhotoShop, or Illustrator artwork and implement the UI in Blend for Visual Studio.	Insert images, videos, and audio clips

TO	VISUAL STUDIO	BLEND FOR VISUAL STUDIO	MORE INFORMATION
Edit controls, templates, and styles	Requires coding and knowledge of WPF styles and templates.	<p>Turn any image into a control.</p> <p>Use the template editing tools to make changes to controls, styles, and templates with just a few mouse clicks.</p> <p>For example, you can use Blend for Visual Studio style resources to implement common WPF controls (such as buttons, list boxes, scroll bars, menus, etc.), and change their color, style, or underlying template directly in Blend for Visual Studio. You can then switch to code for finishing touches if you want.</p>	Modify the style of objects
Connect your UI to data	<p>You can create a data source from resources such as SQL Server databases, WCF or web services, objects, or SharePoint lists, and bind the data source to your UI controls.</p> <p>Design-time data must be created by hand for an interactive design experience.</p>	<p>Create sample data easily for prototyping and testing. Switch to live data when you're ready.</p> <p>Blend for Visual Studio's data generation capabilities are outstanding (you can add names, numbers, URLs, and photos easily on the fly), and can save you a lot of time.</p> <p>For live data, you can bind your UI controls to an XML file or to any CLR data source.</p>	Display data

For more information about advanced XAML design, see [Create a UI by using Blend for Visual Studio](#).

Create a UI by using XAML Designer in Visual Studio

2/8/2019 • 9 minutes to read • [Edit Online](#)

The XAML Designer in Visual Studio provides a visual interface to help you design XAML-based Windows and Web apps. You can create user interfaces for your apps by dragging controls from the **Toolbox** and setting properties in the **Properties** window. You can also edit XAML directly in XAML view.

For advanced XAML design tasks such as animations and behaviors, see [Creating a UI by using Blend for Visual Studio](#). Also see [Design XAML in Visual Studio and Blend for Visual Studio](#) for a comparison between the tools.

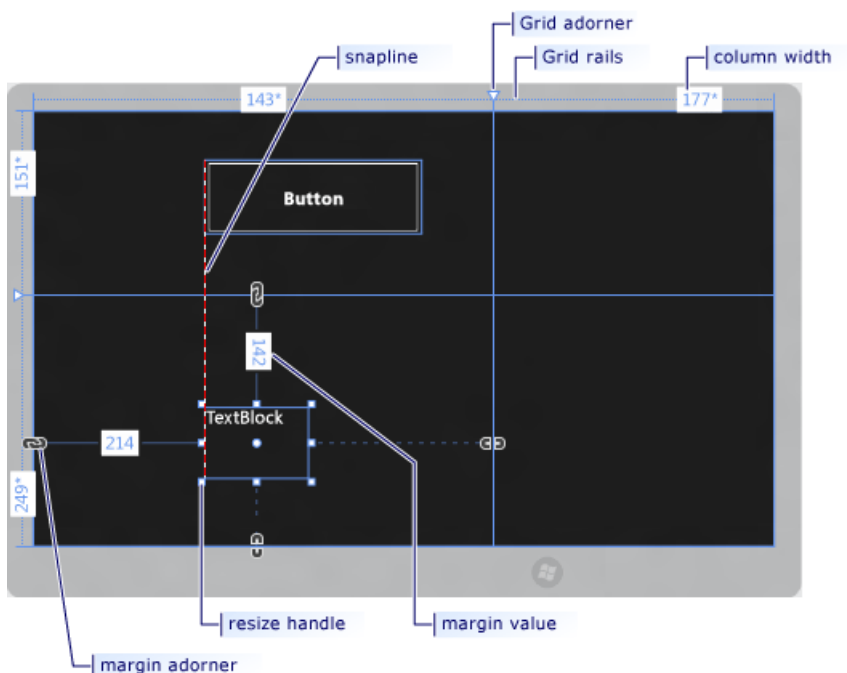
XAML Designer workspace

The workspace in XAML Designer consists of several visual interface elements. These include the **artboard**, **XAML Editor**, **Device** window, **Document Outline** window, and **Properties** window. To open the XAML Designer, right-click a XAML file in **Solution Explorer** and choose **View Designer**.

Authoring views

XAML Designer provides a XAML view and a synchronized Design view of your app's rendered XAML markup. With a XAML file open in Visual Studio, you can switch between Design view and XAML view by using the **Design** and **XAML** tabs. You can use the **Swap Panes** button to switch which window appears on top: either the artboard or the XAML Editor.

In Design view, the window containing the *artboard* is the active window and you can use it as a primary work surface. You can use it to visually design a page in your app by adding or drawing elements, and then by modifying them. For more info, see [Working with elements in XAML Designer](#). This illustration shows the artboard in Design view.



These features are available in the artboard:

Snaplines

Snaplines are *alignment boundaries* that appear as red-dashed lines to show when the edges of controls are aligned, or when text baselines are aligned. Alignment boundaries appear only when **snapping to snaplines** is enabled.

Grid rails

Grid rails are used to manage rows and columns in a **Grid** panel. You can create and delete rows and columns, and you can adjust their relative widths and heights. The vertical Grid rail, which appears on the left of the artboard, is used for rows, and the horizontal line, which appears at the top, is used for columns.

Grid adorners

A Grid adorer appears as a triangle that has a vertical or horizontal line attached to it on the Grid rail. When you drag a Grid adorer, the widths or heights of adjacent columns or rows update as you move the mouse.

Grid adorners are used to control the width and height of a Grid's rows and columns. You can add a new column or row by clicking in the Grid rails. When you add a new row or column line for a Grid panel that has two or more columns or rows, a mini-toolbar appears outside of the rail that enables you to set width and height explicitly. The mini-toolbar enables you to set sizing options for Grid rows and columns.

Resize handles

Resize handles appear on selected controls and enable you to resize the control. When you resize a control, width and height values typically appear to help you size the control. For more information about manipulating controls in **Design** view, see [Working with elements in XAML Designer](#).

Margins

Margins represent the amount of fixed space between the edge of a control and the edge of its container. You can set the margins of a control by using the **Margin** properties under **Layout** in the Properties window.

Margin adorners

You can use margin adorners to change the margins of an element relative to its layout container. When a margin adorer is open, a margin is not set and the margin adorer displays a broken chain. When the margin is not set, elements remain in place when the layout container is resized at run time. When a margin adorer is closed, a margin adorer displays an unbroken chain, and elements move with the margin as the layout container is resized at run time (the margin remains fixed).

Element handles

You can modify an element by using the element handles that appear on the artboard when you move the pointer over the corners of the blue box that surrounds an element. These handles enable you to rotate, resize, flip, move, or add a corner radius to the element. The symbol for the element handle varies by function, and changes depending on the exact location of the pointer. If you don't see the element handles, make sure the element is selected.

In **Design** view, additional artboard commands are available in the lower-left area of the screen, as shown here:



These commands are available on this toolbar:

Zoom

Zoom enables you to size the design surface. You can zoom from 12.5% to 800%, or select options such as **Fit to Selection** and **Fit to All**.

Show/Hide snap grid

Displays or hides the snap grid that shows the gridlines. Gridlines are used when you enable either **snapping to gridlines** or **snapping to snaplines**.

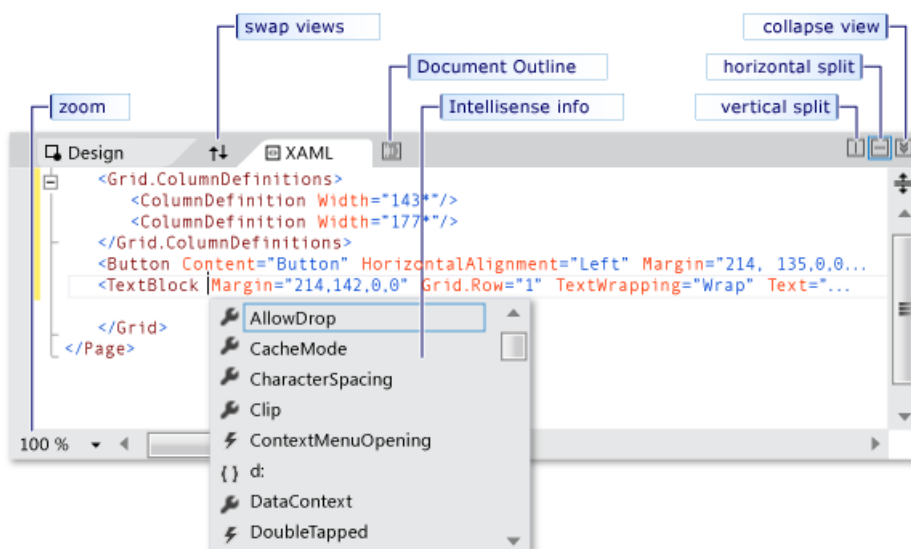
Turn on/off snapping to gridlines

If **snapping to gridlines** is enabled, when you drag an element on the artboard, the element tends to align with the closest horizontal and vertical gridlines.

Turn on/off snapping to snaplines

Snaplines help you align controls relative to each other. If **snapping to snaplines** is enabled, when you drag a control relative to other controls, alignment boundaries appear when the edges and the text of some controls are aligned horizontally or vertically. An alignment boundary appears as a red-dashed line.

In **XAML** view, the window containing the XAML editor is the active window, and the XAML editor is your primary authoring tool. The Extensible Application Markup Language (XAML) provides a declarative, XML-based vocabulary for specifying an application's user interface. XAML view includes IntelliSense, automatic formatting, syntax highlighting, and tag navigation. This illustration shows XAML view:



Split view bar

The split view bar appears at the top of XAML view when the XAML editor is in the lower window. The split view bar enables you to control the relative sizes of **Design** view and **XAML** view. You can also exchange the locations of the views (using the **Swap Panes** button), specify whether the views are arranged horizontally or vertically, and collapse either view.

Markup Zoom

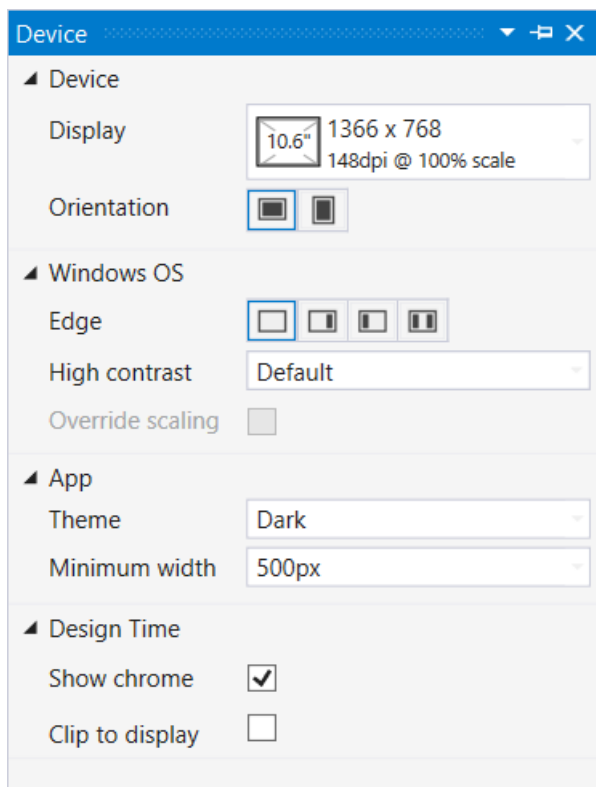
Markup zoom enables you to size **XAML** view. You can zoom from 20% to 400%.

Device window

NOTE

If the target platform version (`TargetPlatformVersion`) of a UWP application is 10.0.16299.0 or higher, the **Device** window is not available.

The **Device** window in XAML Designer enables you to simulate at design-time various views, displays, and display options for your project. The **Device** window is available on the **Design** menu when you are working in the XAML Designer. Here's what it looks like:



These are the options available in the Device window:

Display

Specifies different display sizes and resolutions for the app.

Orientation

Specifies different orientations for the app: **Landscape** or **Portrait**.

Edge

Specifies different edge alignments for your app: **Both**, **Left**, **Right**, or **None**.

High Contrast

Preview the app based on the selected contrast setting. This setting, when set to a value other than **Default**, overrides the `RequestedTheme` property set in *App.xaml*.

Override scaling

Turns on and off emulation of document scaling within the design surface. This enables you to increase the scaling percentage by one factor. Select the check box to turn on emulation. For instance, if your scaling percentage is 100%, the document within the design surface will scale up to 140%. This option is disabled if the current scaling percentage is 180.

Minimum width

Specifies the minimum width setting. The minimum width can be changed in *App.xaml*.

Theme

Specifies the app theme. For example, you might switch between a **Dark** and a **Light** theme.

Show chrome

Turns on and off the simulated tablet frame around your app in Design view. Select the check box to show the frame.

Clip to display

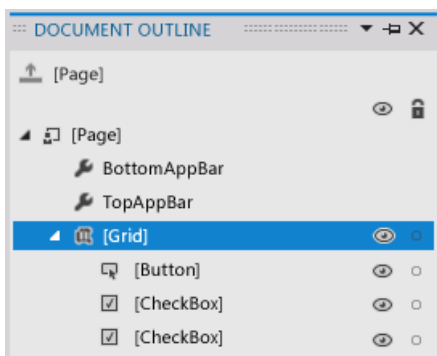
Specifies the display mode. Select the check box to clip the document size to the display size.

Document Outline window

The Document Outline window in XAML Designer helps you perform these tasks:

- View the hierarchical structure of all elements on the artboard.
- Select elements so that you can modify them (move them around in the hierarchy, modify them on the artboard, set their properties in the Properties window, and so on). For more information, see [Working with elements in XAML Designer](#)
- Create and modify templates for elements that are controls.
- Use the right-click menu (context menu) for selected elements. The same menu is also available for selected elements in the artboard.

To view the **Document Outline** window, on the menu bar choose **View > Other Windows > Document Outline**.



These are the options available in the **Document Outline** window:

Document Outline

The main view in the **Document Outline** window displays the hierarchy of a document in a tree structure. You can use the hierarchical nature of the document outline to examine the document at varying levels of detail, and to lock and hide elements singly or in groups.

Show/hide

Displays or hides artboard elements that correspond to items in the document outline. Use the **Show/hide** buttons, which display a symbol of an eye when shown, or press **Ctrl+H** to hide elements and **Shift+Ctrl+H** to display them.

Lock/unlock

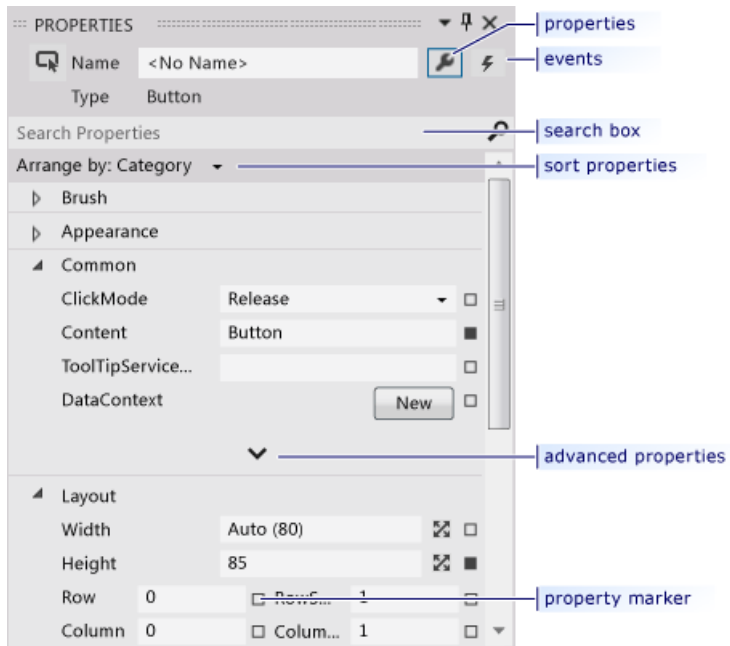
Locks or unlocks artboard elements that correspond to items in the Document Outline. Locked elements can't be modified. Use the **Lock/unlock** buttons, which display a padlock symbol when locked, or press **Ctrl+L** to lock elements and **Shift+Ctrl+L** to unlock them.

Return scope to pageRoot

The option at the top of the **Document Outline** window, which shows an up arrow symbol, returns the document outline to the previous scope. Scoping up is applicable only when you're in the scope of a style or template.

Properties window

The **Properties** window enables you to set property values on controls. Here's what it looks like:



There are various options at the top of the **Properties** window. You can change the name of the currently selected element by using the **Name** box. In the upper-left corner, there's an icon that represents the currently selected element. To arrange the properties by category or alphabetically, click **Category**, **Name**, or **Source** in the **Arrange by** list. To see the list of events for a control, click the **Events** button, which displays a lightning bolt symbol. To search for a property, start to type the name of the property in the **Search Properties** box. The **Properties** window displays the properties that match your search as you type. Some properties allow you to set advanced properties by selecting a down arrow button. For more information on using properties and handling events, see [Intro to controls and patterns](#)

To the right of each property value is a *property marker* that appears as a box symbol. The appearance of the property marker indicates whether there's a data binding or a resource applied to the property. For example, a white box symbol indicates a default value, a black box symbol typically indicates that a local resource has been applied, and an orange box typically indicates a data binding has been applied. When you click the property marker, you can navigate to the definition of a style, open the data binding builder, or open the resource picker.

See also

- [Working with elements in XAML Designer](#)
- [How to create and apply a resource](#)
- [Walkthrough: Binding to data in XAML Designer](#)

Work with elements in XAML Designer

2/8/2019 • 6 minutes to read • [Edit Online](#)

You can add elements—controls, layouts, and shapes—to your app in XAML, in code, or by using XAML Designer. This topic describes how to work with elements in XAML Designer in Visual Studio or Blend for Visual Studio.

Add an element to a layout

Layout is the process of sizing and positioning elements in a UI. To position visual elements, you must put them in a layout **Panel**. A **Panel** has a child property which is a collection of **FrameworkElement** types. You can use various **Panel** child elements, such as **Canvas**, **StackPanel**, and **Grid**, to serve as layout containers and to position and arrange the elements on a page.

By default, a **Grid** panel is used as the top-level layout container within a page or form. You can add layout panels, controls, or other elements within the top-level page layout.

To add an element to a layout in XAML Designer, do one of the following:

- Double-click an element in the **Toolbox** (or select an element in the Toolbox and press **Enter**).
- Drag an element from the **Toolbox** to the artboard.
- In the **Toolbox**, select one of the drawing tools (for example, **Ellipse** or **Rectangle**), and then draw an element in the active panel.

Change the layering order of elements

When there are two elements on the artboard in XAML Designer, one element will appear in front of the other in the layering order. At the bottom of the list of elements, in the Document Outline window is the front-most element (except for when the **ZIndex** property for an element is set). When you insert an element into a page, form, or layout container, the element is automatically placed in front of other elements in the active container element. To change the order of elements, you can use the **Order** commands or drag the elements in the object tree in the Document Outline window.

To change the layering order, do one of the following:

- In the **Document Outline** window, drag the elements up or down to create the desired layering order.
- Right-click the element in the Document Outline window or the artboard for which you want to change the layering order, point to **Order**, and then click one of the following:
 - **Bring to Front** to bring the element all the way to the front of the order.
 - **Bring Forward** to bring the element forward one level in the order.
 - **Send Backward** to send the element back one level in the order.
 - **Send to Back** to send the element all the way to the back of the order.

Change the **ZIndex** property in the **Layout** section in the Properties window. For overlapping elements, the **ZIndex** property takes precedence over the order of elements shown in the Document Outline window. An element that has a higher **ZIndex** value appears in front when elements overlap.

Change the alignment of an element

You can align elements in the artboard by using menu commands or by dragging elements to snaplines.

A *snapline* is a visual cue that helps you align an element relative to other elements in the app.

To align two or more elements by using menu commands:

1. Select the elements that you want to align. You can select more than one element by pressing and holding the **Ctrl** key while you select the elements.
2. Select one of the following properties under **HorizontalAlignment** in the **Layout** section of the Properties window: **Left**, **Center**, **Right**, or **Stretch**.
3. Select one of the following properties under **VerticalAlignment** in the **Layout** section of the Properties window: **Top**, **Center**, **Bottom**, or **Stretch**.

To align two or more elements by using snaplines, in XAML Designer, in a layout that contains at least two elements, drag or resize one of the elements so that the edge is aligned with another element.

When the edges are aligned, an *alignment boundary* appears to indicate alignment. The alignment boundary is a red dashed line. Alignment boundaries appear only when **snapping to snaplines** is enabled. For an illustration of the artboard that shows an alignment boundary, see [Creating a UI by using XAML Designer](#).

Change an element's margins

The margins in XAML Designer determine the amount of empty space that is around an element on the artboard. For example, margins specify the amount of space between the outside edges of an element and the boundaries of a `Grid` panel that contains the element. Margins also specify the amount of space between elements that are contained in a `StackPanel`.

To change an element's margins in the Properties window:

1. Select the element whose margins you want to change.
2. Under **Layout** in the Properties window, change the value (in pixels or device-independent units, which are approximately 1/96 inch) for any of the **Margin** properties (**Top**, **Left**, **Right**, or **Bottom**).

In the artboard, to change an element's margins relative to the element's layout container, click the *margin adorners* that appear around the element when the element is selected and is within a layout container. For an illustration that shows margin adorners, see [Creating a UI by using XAML Designer](#).

If a margin adorer is open, vertically or horizontally, that margin isn't set. If a margin adorer is closed, that margin is set.

When you open a margin adorer and the opposite margin isn't set, the opposite margin is set to the correct value according to the location of the element in the artboard. For opposite margins, such as the **Left** and **Right** margins, at least one property is always set.

IMPORTANT

Elements placed inside some layout containers, such as a `Windows.UI.Xaml.Controls.Canvas`, don't have margin adorners. Elements placed inside a `Windows.UI.Xaml.Controls.StackPanel` have margin adorners for either the left and right margins or the top and bottom margins, depending on the orientation of the `StackPanel`.

Group and ungroup elements

Grouping two or more elements in XAML Designer creates a new layout container and places those elements within that container. Placing two or more elements together in a layout container enables you to easily select,

move, and transform the group as if the elements in that group were one element. Grouping is also useful for identifying elements that are related to each other in some way, such as the buttons that make up a navigation element. When you ungroup elements, you are simply deleting the layout container that contained the elements.

To group elements into a new layout container:

1. Select the elements that you want to group. (To select multiple elements, press and hold the **Ctrl** key while you click them.)
2. Right-click the selected elements, point to **Group Into**, and then click the type of layout container in which you want the group to reside.

TIP

If you select `Windows.UI.Xaml.Controls.Viewbox`, `Windows.UI.Xaml.Controls.Border`, or `Windows.UI.Xaml.Controls.ScrollViewer` to group your elements, the elements are placed in a new `Windows.UI.Xaml.Controls.Grid` panel within the `Windows.UI.Xaml.Controls.Viewbox`, `Windows.UI.Xaml.Controls.Border`, or `Windows.UI.Xaml.Controls.ScrollViewer`. If you ungroup elements in one of these layout containers, only the `Windows.UI.Xaml.Controls.Viewbox`, `Windows.UI.Xaml.Controls.Border`, or `Windows.UI.Xaml.Controls.ScrollViewer` is deleted, and the `Windows.UI.Xaml.Controls.Grid` panel remains. To delete the `Grid` panel, ungroup the elements again.

To ungroup elements and delete the layout, right-click the group that you want to ungroup and click **Ungroup**. You can also group or ungroup elements by right-clicking selected items in the Document Outline window and clicking **Group Into** or **Ungroup**.

Reset the element layout

You can restore default values for specific layout properties of an element by using the Layout Reset commands. By using this command, you can reset the margin, alignment, width, height, and size of an element, either individually or collectively.

To reset the element layout, right-click the element in the Document Outline window or the artboard, and then choose **Layout > Reset *PropertyName***, where *PropertyName* is the property that you want to reset (or choose **Layout > Reset All** to reset all the layout properties for the element).

See also

- [Create a UI by using XAML Designer](#)

Organize objects into layout containers in XAML Designer

2/8/2019 • 3 minutes to read • [Edit Online](#)

This article describes layout panels and controls for XAML Designer.

Imagine where you'd like objects to appear on a page; objects such as images, buttons, and videos. Maybe you want them to appear in rows and columns, in a single line vertically or horizontally, or in fixed positions.

After you've had a chance to think about how the page might appear, choose a layout panel. All pages start with one because you need something to which you add your objects. By default it's a **Grid**, but you can change that.

Layout panels help you arrange objects on a page, but they do more than that. They help you design for different screen sizes and resolutions. When users run your app, everything in a layout panel resizes to match the screen real estate of their device. Of course, if you don't want your layout to do that, you can override that behavior for a part of the layout, or the entire layout. You can use height and width properties to control that.

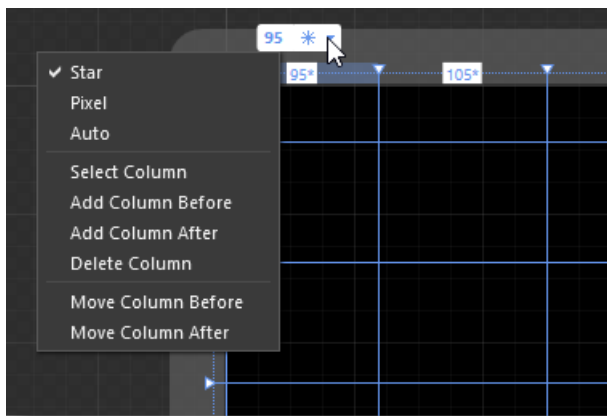
Layout panels

Start your page by choosing one of these layout panels. Your page can have more than one. For example, you might start with a **Grid** layout panel, and then add a **StackPanel** to an area in the **Grid** so that you can arrange controls vertically in that element.

The following layout panels are the most popularly used, but there are others. You can find them all in **Toolbox** in Visual Studio or the **Assets** panel in Blend for Visual Studio.

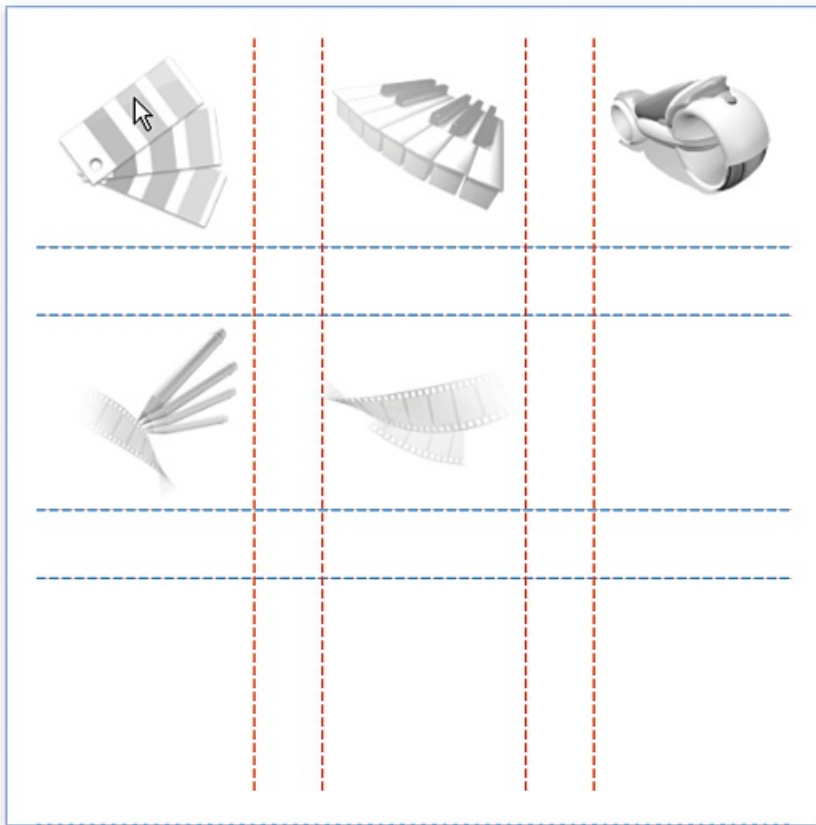
Grid

Arrange objects into rows and columns.



UniformGrid

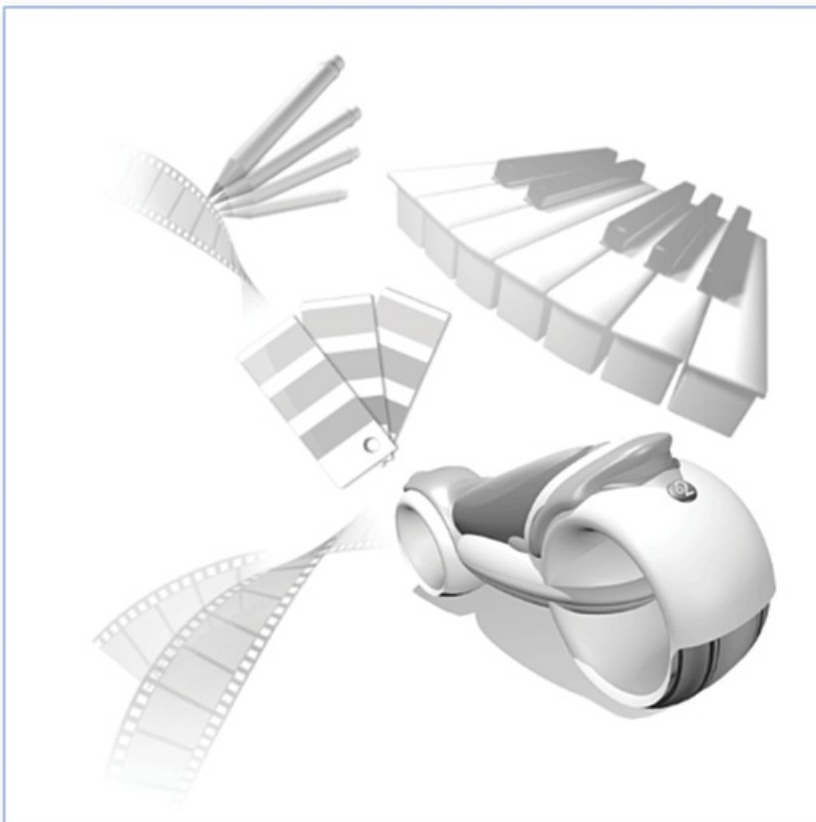
Arrange objects into equal, or uniform, grid regions. This panel is great for arranging a list of images.



(Available only for WPF projects.)

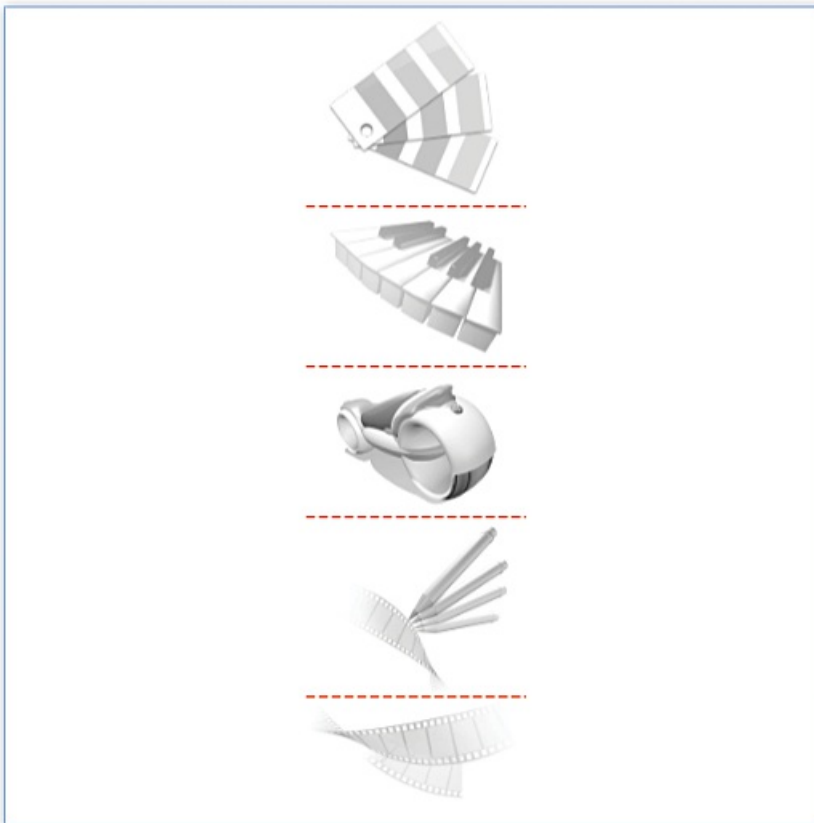
Canvas

Arrange objects any way you want. When users run your app, these elements will have fixed positions on the screen.



StackPanel

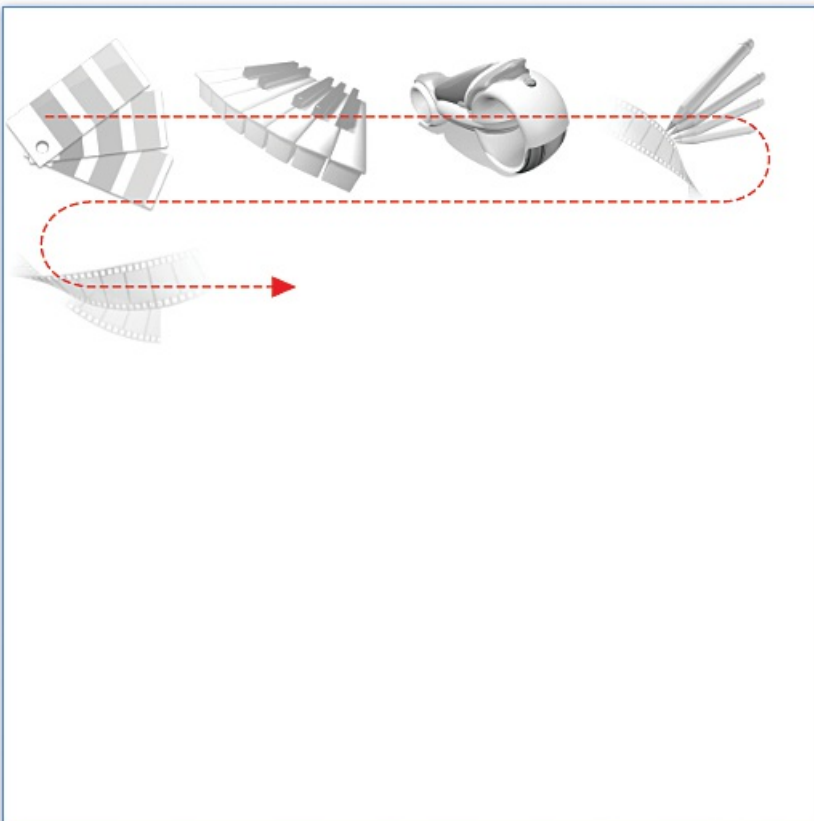
Arrange objects in a single line horizontally or vertically.



WrapPanel

Arrange objects sequentially from left to right. When the panel runs out of room at the far-right edge, it *wraps* the content to the next line, and so on from left-to-right, top-to-bottom. You can also make the orientation of a wrap panel vertical so that objects flow from top-to-bottom, left-to-right.

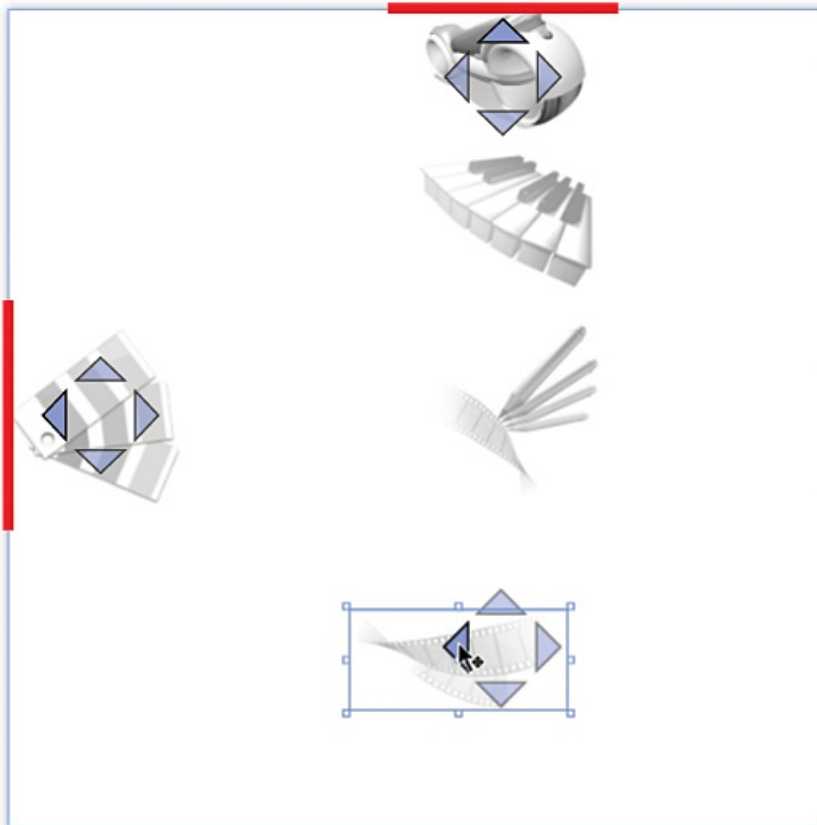
(Available only for WPF projects.)



DockPanel

Arrange objects so that they stay, or *dock*, to one edge of the panel.

(Available only for WPF projects.)



Watch a short video: [WPF - DockPanel](#)

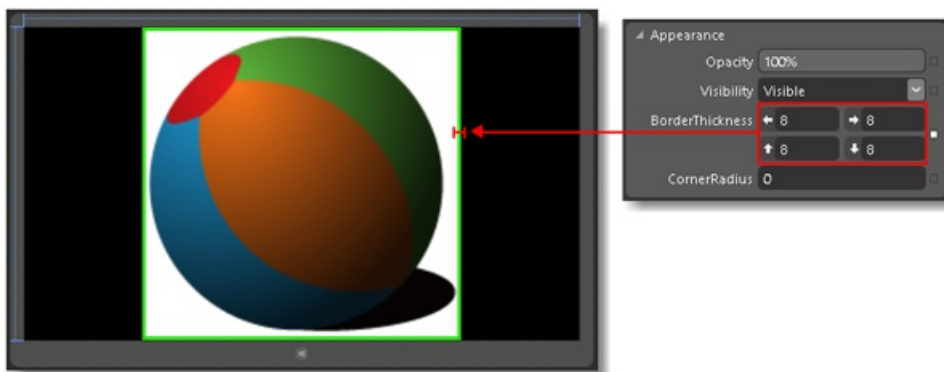
Layout controls

You can add your objects to layout controls as well. They aren't as feature-rich as a layout panel, but you might find them helpful for certain scenarios.

The following layout controls are the most popular, but there are others. You can find them all in **Toolbox** in Visual Studio or the **Assets** panel in Blend for Visual Studio.

Border

Create a border, background, or both around an object. You can add only one object to a **Border**. If you want to apply a border or background for more than one object, add layout panel to the **Border**. Then, add objects to that panel or control.

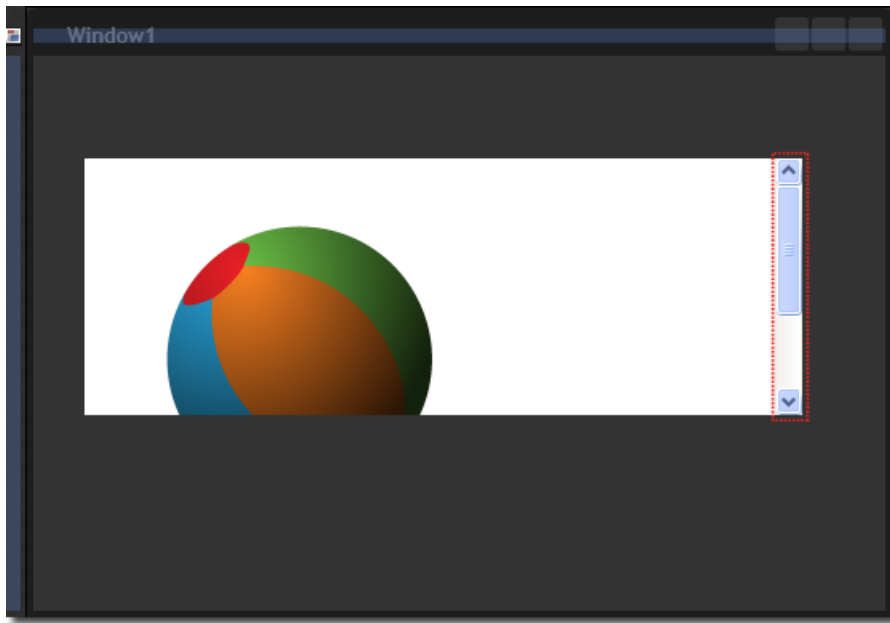


Popup

Show information or options to users in a window. You can add only one object to a **Popup**. By default, a **Popup** contains a **Grid** but you can change that.

ScrollViewer

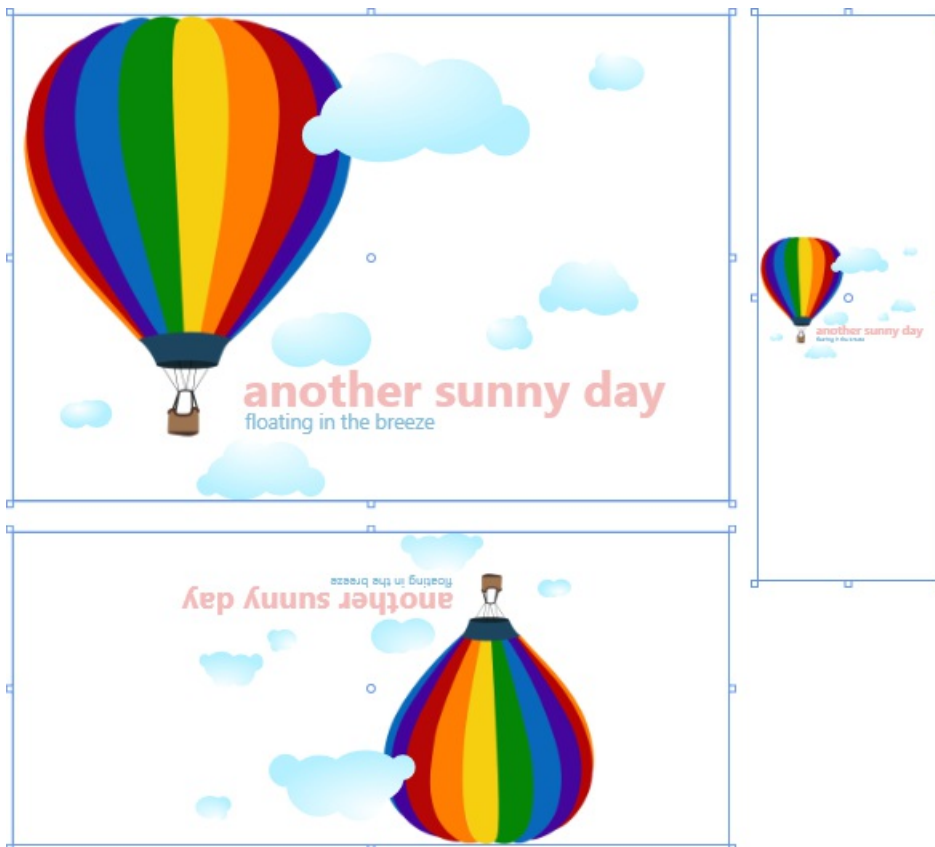
Enable users to scroll down a page or area of a page. You can add only one object to a **ScrollViewer** so it makes a lot of sense to add a layout panel such as a **Grid** or **StackPanel**.



Viewbox

Scale objects much like you would with a zoom control. You can add only one object to a **Viewbox**. If you want to apply that effect to more than one object, add a layout panel to the **ViewBox**, and then add your controls to that layout panel.

(Available only for WPF projects.)



See also

- [Work with elements in XAML Designer](#)

- [Create a UI by using XAML Designer](#)

How to create and apply a resource

2/8/2019 • 3 minutes to read • [Edit Online](#)

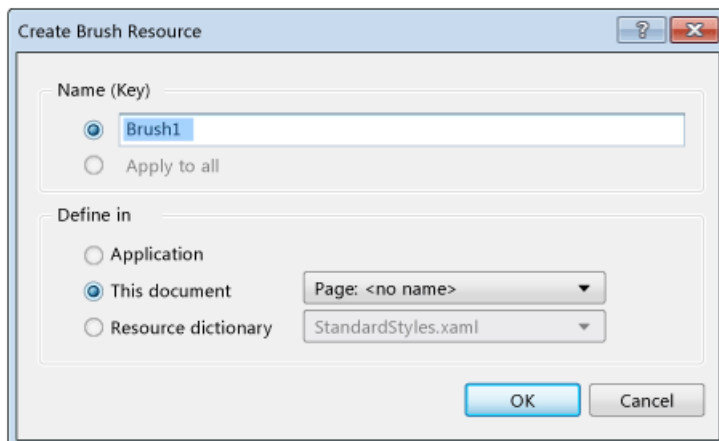
Styles and templates for elements in the XAML Designer are stored in reusable entities called resources. Styles enable you to set element properties and reuse those settings for a consistent appearance across multiple elements. A [ControlTemplate](#) defines the appearance of a control and can also be applied as a resource. For more information, see [Quickstart: Styling controls](#) and [QuickStart: Control templates](#).

Whenever you create a new resource from an existing property, [Style](#), or `ControlTemplate`, the **Create Resource** dialog box enables you to define the resource at the application level, the document level, or the element level. These levels determine where you can use the resource. For example, if you define the resource at the element level, the resource can be applied only to the element on which you created it. You can also choose to store the resource in a resource dictionary, which is a separate file that you can use again in another project.

To create a new resource

1. With a XAML file open in XAML Designer, create an element, or choose an element in the Document Outline window.
2. In the Properties window, choose the property marker, which appears as a box symbol to the right of a property value, and then choose **Convert to New Resource**. A white box symbol indicates a default value, and a black box symbol typically indicates that a local resource has been applied.

The appropriate dialog box for creating a resource appears. This dialog box appears when you create a resource from a brush:



3. In the **Name (Key)** box, enter a key name. This is the name that you can use when you want other elements to reference the resource.
4. Under **Define in**, choose the option that specifies where you want the resource to be defined:
 - To make the resource available to any document in your application, choose **Application**.
 - To make the resource available to only the current document, choose **This document**.
 - To make the resource available to only the element from which you created the resource or to its child elements, choose **This document**, and in the drop-down list, select **element: name**.
 - To define the resource in a resource dictionary file that can be reused in other projects, click **Resource dictionary**, and then select an existing resource dictionary file, such as **StandardStyles.xaml**, in the drop-down list.

5. Choose the **OK** button to create the resource and apply it to the element from which you created it.

To apply a resource to an element or property

1. In the Document Outline window, choose the element to which you want to apply a resource.
2. Do one of the following:
 - Apply a resource to a property. In the Properties window, choose the property marker next to the property value, choose **Local Resource** or **System Resource**, and then choose an available resource from the list that appears.

If you don't see a resource that you expect to see, it might be because the type of the resource doesn't match the type of the property.

- Apply a style or control template resource to a control. Open the right-click menu (context menu) for a control in the Document Outline window, choose **Edit Template** or **Edit Additional Templates**, choose **Apply Resource**, and then choose the name of the control template from the list that appears.

NOTE

Edit Template applies control templates. **Edit Additional Templates** applies other template types.

You can apply resources wherever they're compatible. For example, you can apply a brush resource to the **Foreground** property of a [Windows.UI.Xaml.Controls.TextBox](#) control.

To edit a resource

1. Choose an element on the artboard or in the Document Outline window.
2. Choose the Default or Local property marker to the right of the property in the Properties window, and then choose **Edit Resource** to open the **Edit Resource** dialog box.
3. Modify options for the resource.

See also

- [Creating a UI by using XAML Designer](#)

Walkthrough: Bind to data in XAML Designer

2/8/2019 • 2 minutes to read • [Edit Online](#)

In XAML Designer, you can set data binding properties by using the artboard and the Properties window. The example in this walkthrough shows how to bind data to a control. Specifically, the walkthrough shows how to create a simple shopping cart class that has a [DependencyProperty](#) named `ItemCount`, and then bind the `ItemCount` property to the **Text** property of a [TextBlock](#) control.

To create a class to use as a data source

1. On the **File** menu, choose **New > Project**.
2. In the **New Project** dialog box, choose either the **Visual C#** or **Visual Basic** node, expand the **Windows Desktop** node, and then choose the **WPF Application** template.
3. Name the project **BindingTest**, and then choose the **OK** button.
4. Open the **MainWindow.xaml.cs** (or **MainWindow.xaml.vb**) file and add the following code. In C#, add the code in the `BindingTest` namespace (before the final closing parenthesis in the file). In Visual Basic, just add the new class.

```
public class ShoppingCart : DependencyObject
{
    public int ItemCount
    {
        get { return (int)GetValue(ItemCountProperty); }
        set { SetValue(ItemCountProperty, value); }
    }

    public static readonly DependencyProperty ItemCountProperty =
        DependencyProperty.Register("ItemCount", typeof(int),
            typeof(ShoppingCart), new PropertyMetadata(0));
}
```

```
Public Class ShoppingCart
    Inherits DependencyObject

    Public Shared ReadOnly ItemCountProperty As DependencyProperty = DependencyProperty.Register(
        "ItemCount", GetType(Integer), GetType(ShoppingCart), New PropertyMetadata(0))
    Public Property ItemCount As Integer
        Get
            ItemCount = CType(GetValue(ItemCountProperty), Integer)
        End Get
        Set(value As Integer)
            SetValue(ItemCountProperty, value)
        End Set
    End Property
End Class
```

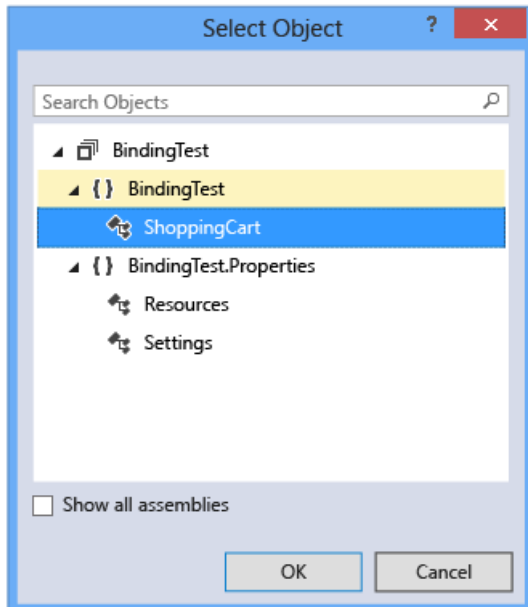
This code sets a value of 0 as the default item count by using the [PropertyMetadata](#) object.

5. On the **File** menu, choose **Build > Build Solution**.

To bind the ItemCount property to a TextBlock control

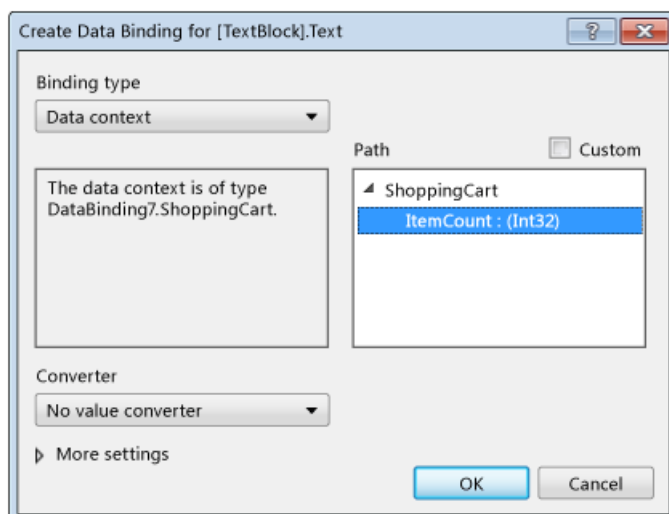
1. In Solution Explorer, open the shortcut menu for **MainWindow.xaml** and choose **View Designer**.
2. In the Toolbox, choose a **Grid** control and add it to the form.
3. With the **Grid** selected, in the Properties window, choose the **New** button next to the **DataContext** property.
4. In the **Select Object** dialog box, make sure that **Show all assemblies** checkbox is cleared, choose **ShoppingCart** under the **BindingTest** namespace, and then choose the **OK** button.

The following illustration shows the **Select Object** dialog box with **ShoppingCart** selected.



5. In the **Toolbox**, choose a **TextBlock** control to add it to the form.
6. With the **TextBlock** control selected, in the Properties window, choose the property marker to the right of the **Text** property, and then choose **Create Data Binding**. (The property marker looks like a small box.)
7. In the Create Data Binding dialog box, in the **Path** box, choose the **ItemCount : (int32)** property and then choose the **OK** button.

The following illustration shows the **Create Data Binding** dialog box with the **ItemCount** property selected.



8. Press **F5** to run the app.

The **TextBlock** control should show the default value of 0 as text.

See also

- [Create a UI by using XAML Designer](#)
- [Add Value Converter dialog box](#)

Keyboard shortcuts for XAML Designer

2/8/2019 • 2 minutes to read • [Edit Online](#)

Keyboard shortcuts in XAML Designer can speed up your work by reducing an action that would require multiple mouse-button clicks to a single keyboard shortcut.

Element shortcuts

This table lists the shortcuts that are available for working with elements on the artboard.

TO PERFORM THIS ACTION	DO THIS
Create an element	Press Ctrl+N
Duplicate an element	Hold down Alt and press an arrow key.
Edit the text in a control	Press F2 (press Esc to exit)
Select a single element	Press Tab to select elements in the order they appear in the document. (You can also select elements using the arrow keys.)
Select multiple elements	Press and hold Shift while selecting each element
Select multiple non-adjacent elements	Press and hold Ctrl while selecting the first and last elements
Move selected elements	Press the arrow keys (You can hold down Shift to increase the distance to move per key press.)
Rotate an element in 15-degree increments	Hold down Shift while rotating the element
Select all elements	Press Ctrl+A
Clear selection of all elements	Press Ctrl+Shift+A
Show or hide element handles	Press F9
Select a property for an element	With an element selected and the focus on the Properties window, press Tab . (Use Ctrl+Tab to change focus to the Properties window.) You can use the arrow keys to select property values from drop-down lists.

Document Outline window shortcuts

The following table lists the shortcuts available when working with elements in the Document Outline window.

TO PERFORM THIS ACTION	DO THIS
Hide artboard objects while focus is on the Document Outline window	Ctrl+H

TO PERFORM THIS ACTION	DO THIS
Unhide artboard objects while focus is on the Document Outline window	Shift+Ctrl+H
Lock artboard objects while focus is on the Document Outline window	Ctrl+L
Unlock artboard objects while focus is on the Document Outline window	Shift+Ctrl+L

See also

- [Creating a UI by using XAML Designer](#)

Debug or disable project code in XAML Designer

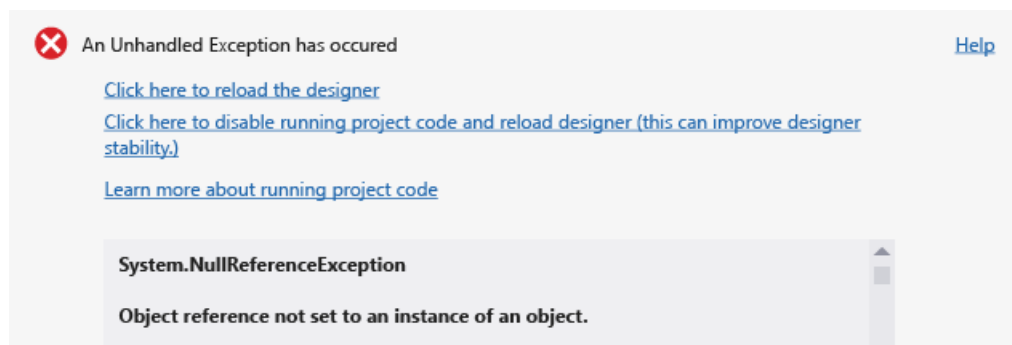
2/8/2019 • 3 minutes to read • [Edit Online](#)

In many cases, unhandled exceptions in the **XAML** designer can be caused by project code attempting to access properties or methods that return different values, or work in a different way when your application runs in the designer. You can resolve these exceptions by debugging the project code in another instance of Visual Studio, or temporarily prevent exceptions by disabling project code in the designer.

Project code includes:

- Custom controls and user controls
- Class libraries
- Value converters
- Bindings against design time data generated from project code

When project code is disabled, Visual Studio shows placeholders. For example, Visual Studio shows the name of the property for a binding where the data is no longer available, or a placeholder for a control that's no longer running.



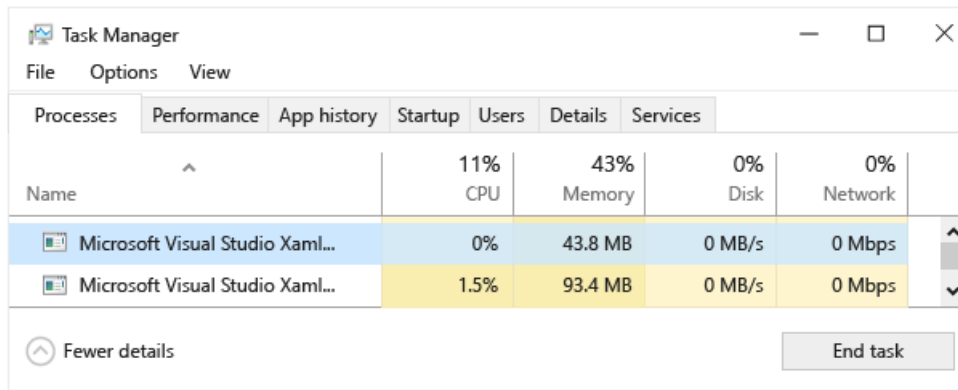
To determine if project code is causing an exception

1. In the unhandled exception dialog, choose the **Click here to reload the designer** link.
2. On the menu bar choose **Debug > Start Debugging** to build and run the application.

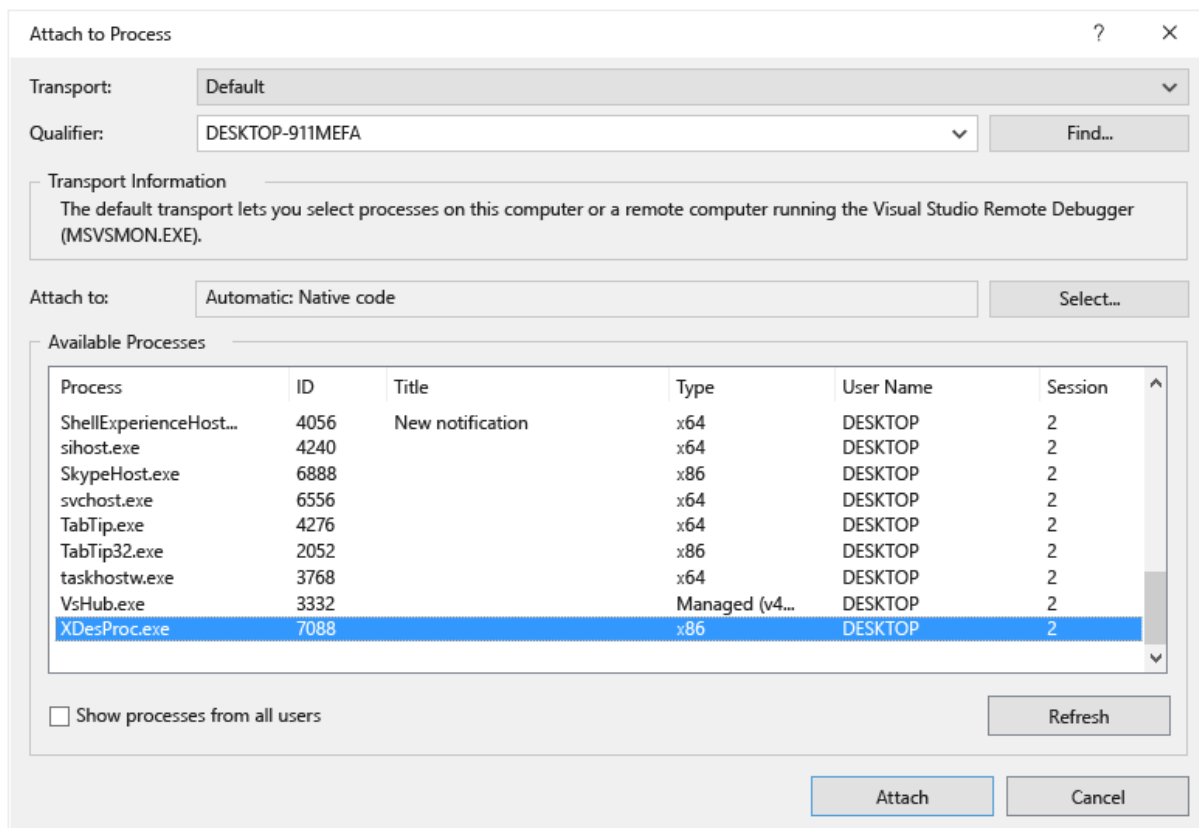
If the application builds and runs successfully, the design-time exception may be caused by your project code running in the designer.

To debug project code running in the designer

1. In the unhandled exception dialog, choose the **Click here to disable running project code and reload designer** link.
2. In the Windows Task Manager, choose the **End Task** button to close any instances of the Visual Studio XAML Designer that are currently running.



3. In Visual Studio, open the XAML page which contains the code or control you want to debug.
4. Open a new instance of Visual Studio, and then open a second instance of your project.
5. Set a breakpoint in your project code.
6. In the new instance of Visual Studio, on the menu bar, choose **Debug** > **Attach to Process**.
7. In the **Attach to Process** dialog, in the **Available Processes** list, choose **XDesProc.exe**, and then choose the **Attach** button.



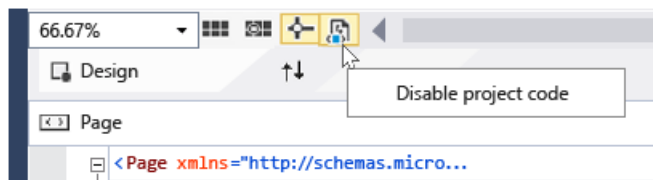
This is the process for the XAML designer in the first instance of Visual Studio.

8. In the first instance of Visual Studio, on the menu bar, choose **Debug** > **Start Debugging**.

You can now step into your code which is running in the designer.

To disable project code in the designer

- In the unhandled exception dialog, choose the **Click here to disable running project code and reload designer** link.
- Alternatively, on the toolbar in the **XAML designer**, choose the **Disable project code** button.



You can toggle the button again to re-enable project code.

NOTE

For projects that target ARM or X64 processors, Visual Studio cannot run project code in the designer, so the **Disable project code** button is disabled in the designer.

- Either option causes the designer to reload and then disable all code for the associated project.

NOTE

Disabling project code can lead to a loss of design-time data. An alternative is to debug the code running in the designer.

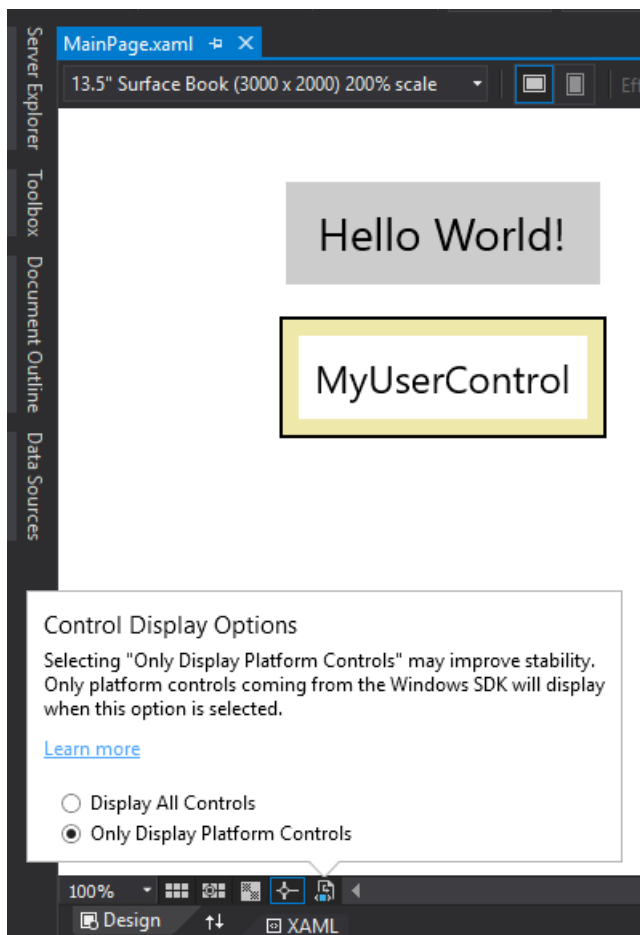
Control display options

NOTE

Control Display Options is only available for Universal Windows Platform applications that target the Windows 10 Fall Creators Update (build 16299) or later. The **Control Display Options** feature is available in Visual Studio 2017 version 15.9 or later.

In the XAML designer, you can change your control display options to only display platform controls from the Windows SDK. This may improve reliability of the XAML designer.

To change control display options, click the icon in the bottom left of the designer window, and then select an option under **Control Display Options**:



When you select **Only Display Platform Controls**, all custom controls coming from SDKs, customer user controls, and more, will not render completely. Instead, they are replaced by fallback controls to demonstrate the size and position of the control.

See also

- [Design XAML in Visual Studio and Blend for Visual Studio](#)

XAML errors and warnings

2/8/2019 • 2 minutes to read • [Edit Online](#)

When authoring XAML, Visual Studio analyzes the code as you type. A squiggle appears on a line of code when an error is detected. Hovering over the squiggle gives you more information about the error or warning. For some errors and warnings, a Quick Action lightbulb is displayed, and using the **Ctrl+.** keyboard shortcut displays the options to fix the issue.

Error types

Behind the scenes, multiple tools analyze the XAML in parallel. XAML errors are categorized into one of the following three types, based on the tool that detected the error:

ERROR DETECTED BY	ERROR CODE FORMAT
XAML Language Service (XAML editor)	XLSxxxx
XAML Designer	XDGxxxx
XAML Edit and Continue	XECxxxx

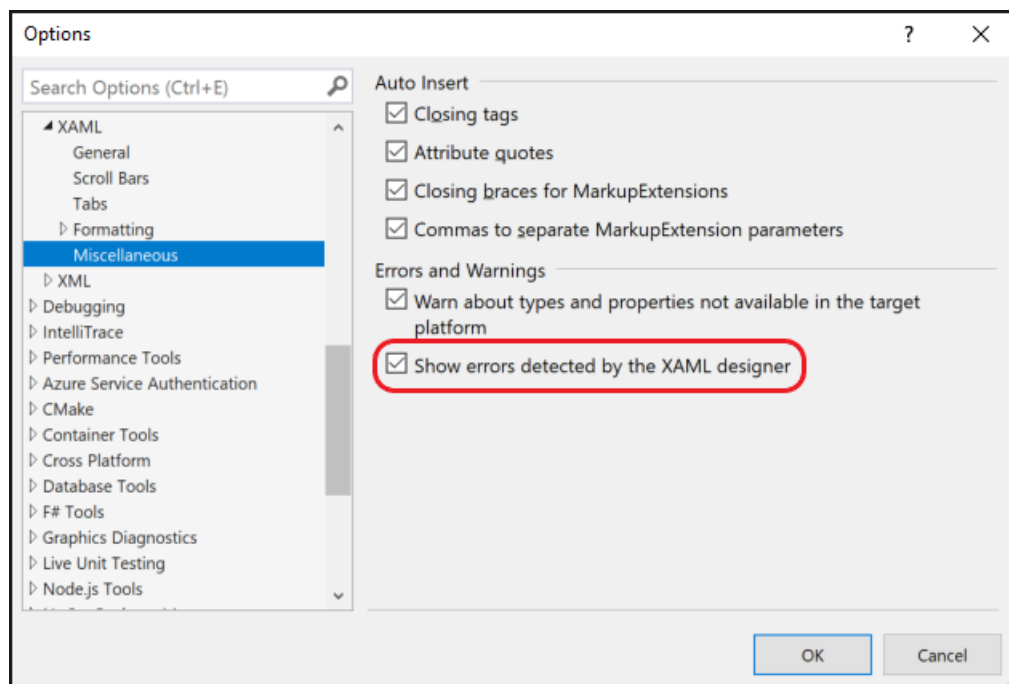
NOTE

Not all errors or warnings have a corresponding code. Such errors are usually XAML Designer errors.

Suppress XAML Designer errors

Open the **Options** dialog by selecting **Tools > Options**, and then select **Text Editor > XAML > Miscellaneous**.

Uncheck the **Show errors detected by the XAML designer** check box.



Blend for Visual Studio overview

2/8/2019 • 5 minutes to read • [Edit Online](#)

Blend for Visual Studio helps you design XAML-based Windows and Web applications. It provides the same basic XAML design experience as Visual Studio and adds visual designers for advanced tasks such as animations and behaviors. For a comparison between Blend and Visual Studio, see [Design XAML in Visual Studio and Blend for Visual Studio](#).

Blend for Visual Studio is a component of Visual Studio. To install Blend, in the **Visual Studio Installer** choose either the **Universal Windows Platform development** or **.NET desktop development** workload. Both of these workloads include the Blend for Visual Studio component.

Summary

- > Visual Studio core editor
- ✓ Universal Windows Platform development
 - Included
 - ✓ Blend for Visual Studio
 - ✓ .NET Native and .NET Standard
 - ✓ NuGet package manager
 - ✓ Universal Windows Platform tools
 - ✓ Windows 10 SDK (10.0.16299.0) for UWP: C#, VB, JS

Summary

- > Visual Studio core editor
- ✓ .NET desktop development
 - Included
 - ✓ .NET desktop development tools
 - ✓ .NET Framework 4.6.1 development tools
 - ✓ C# and Visual Basic
 - Optional
 - ✓ .NET Framework 4 – 4.6 development tools
 - ✓ Blend for Visual Studio
 - ✓ Entity Framework 6 tools
 - ✓ .NET profiling tools
 - ✓ Just-In-Time debugger
 - ☐ F# desktop language support

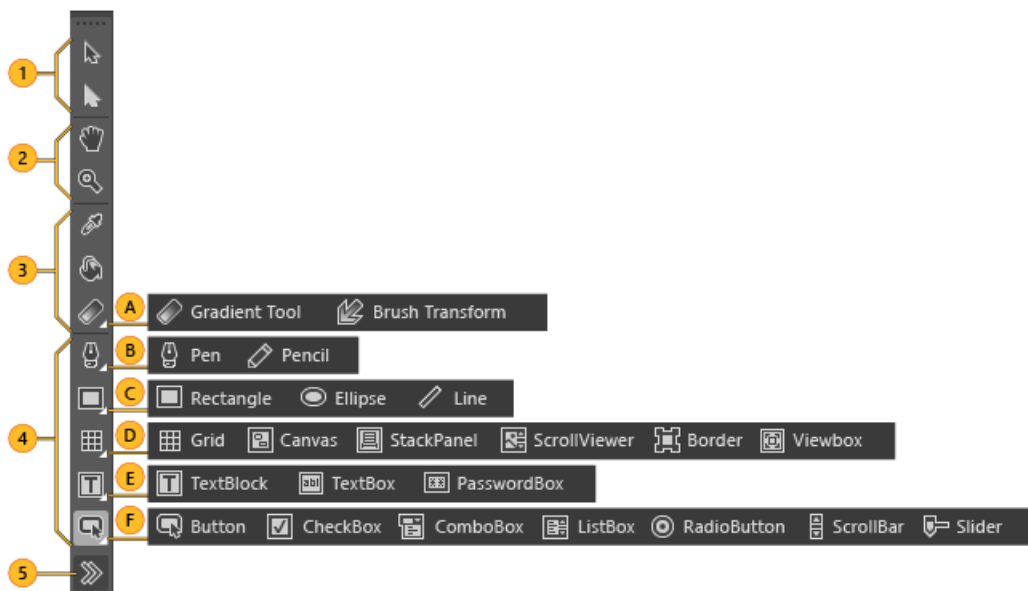
If you're new to Blend for Visual Studio, take a moment to become familiar with the unique features of the workspace. This topic takes you on a quick tour.

NOTE

To tour the shared design features such as the artboard, **Document Outline** window, and **Device** window, see [Creating a UI by using XAML Designer](#).

Tour of the Tools panel

You can use the **Tools** panel in Blend for Visual Studio to create and modify objects in your application. You create the objects by selecting a tool and drawing on the artboard with your mouse.

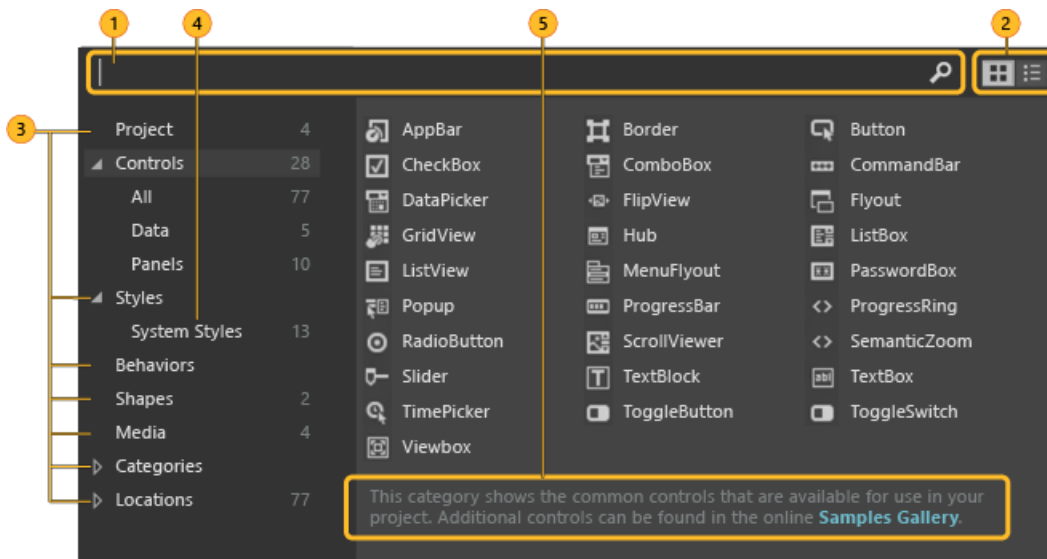


1	Selection tools Select objects and paths. Use the Direct Selection tool to select nested objects and path segments.	A	Gradient and brush tools
2	View tools Adjust the view of the artboard, such as for panning and zooming.	B	Path tools
3	Brush tools Work with the visual attributes of an object, such as transforming a brush, painting an object, or selecting the attributes of one object to apply them to another object.	C	Shape tools
4	Object tools Draw the most common objects on the artboard, such as paths, shapes, layout panels, text, and controls.	D	Layout panels
5	Asset tools Access the Assets panel and to show the most recently used asset from the library.	E	Text controls
		F	Common controls

Watch a short video: [The Toolbar](#).

Tour of the Assets panel

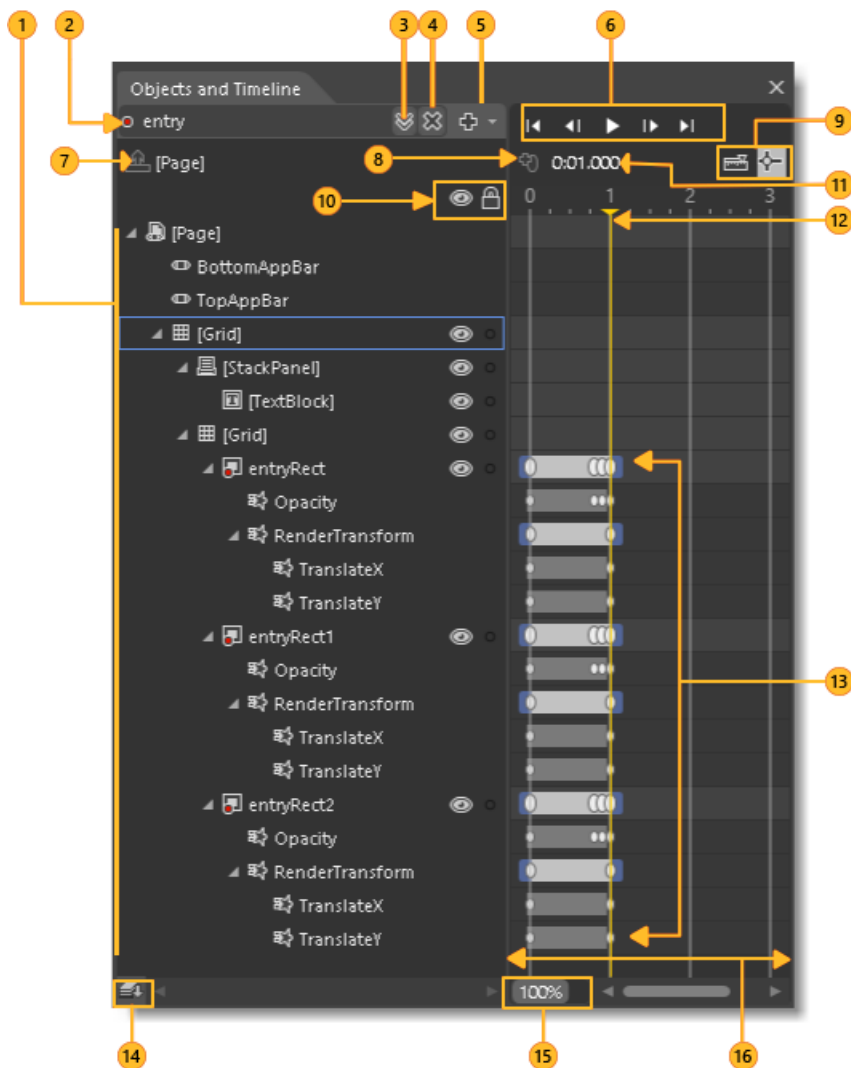
You can find all controls in the **Assets** panel, similar to the **Toolbox** in Visual Studio. In addition to controls, you'll find everything you can add to your artboard in the **Assets** panel, including styles, media, behaviors, and effects.



1	Search box Type in the Search box to filter the list of assets.
2	Grid mode and List mode Switch between the Grid mode view and the List mode view of assets.
3	Assets categories Click a category or subcategory to view the list of assets in that category.
4	Styles Show all the styles that are contained in the resource dictionary.
5	Description View a description of the selected assets category or subcategory.

Tour of the Objects and Timeline panel

Use this panel to organize the objects on your artboard and, if you want, to animate them.

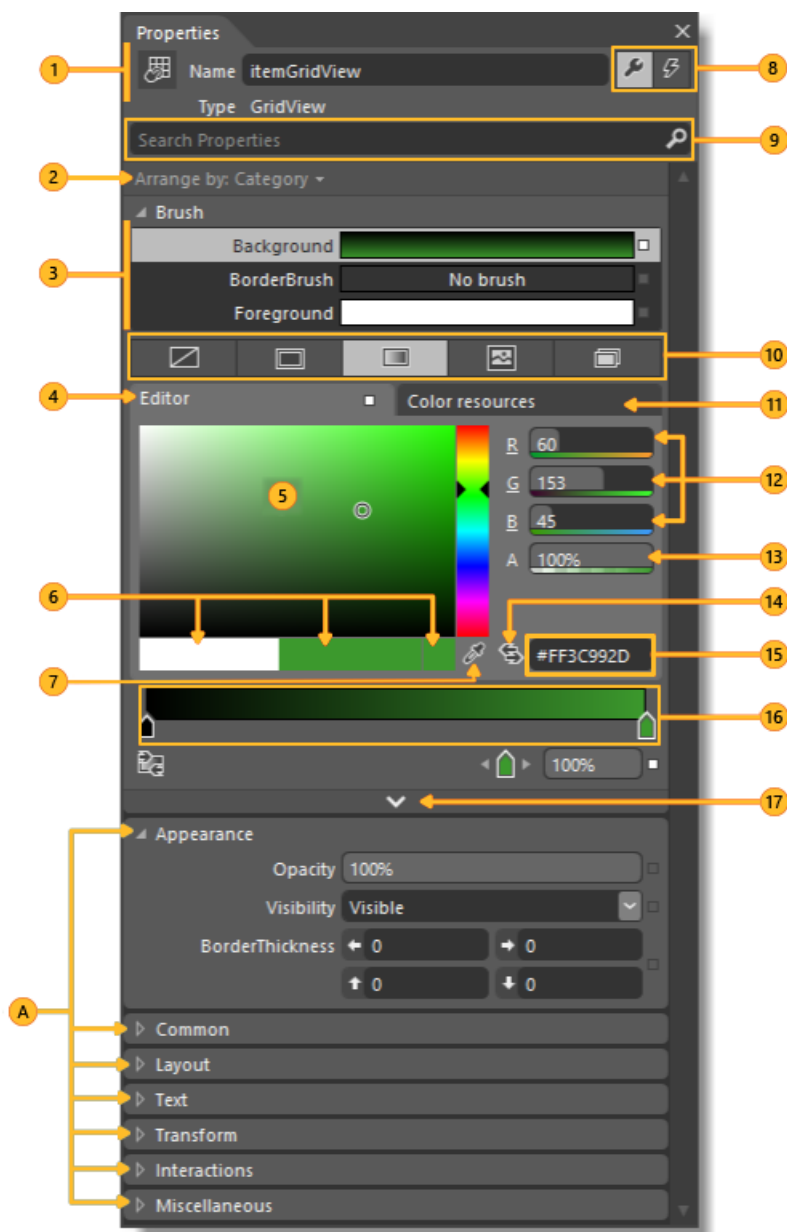




1	Objects view View a visual tree of a document. You can drill down to varying levels of detail. You can also add layers to further organize objects on the artboard. That way you can lock and hide them as a group.
2	Record mode indicator See whether you're recording property changes in a timeline.
3	Storyboard picker View a list of storyboards that you've created.
4	Close storyboard Close the current storyboard.
5	Storyboard options Create, duplicate, reverse, delete, rename, or close a storyboard.
6	Playback controls Navigate through the timeline. You can also drag the playhead to navigate through (or <i>scrub</i>) the timeline.
7	Return scope to Scope the objects view back to the previous root object or previous scope. You can do this only when you're modifying a style or template.

8	Record a keyframe Record a snapshot of the properties of the selected object at the current point in time.
9	Snapping options Set timeline snapping, snap resolution, and turn off timeline snapping.
10	Show/hide, Lock/unlock Show or hide the visibility and locking options for the objects view.
11	Playhead position on the timeline Show the current time in milliseconds. You can also enter a time value directly in this field to jump to a particular point in time. The precision depends on the snap resolution set in the Snapping Options .
12	Playhead Determine what point in time the animation is at. You can drag the playhead across the timeline to preview animation.
13	Keyframes set on timelines Change a property value at a specific point in time.
14	Change order of objects Set the display order of objects. Click this button to arrange objects in the structure view by Z order (front-to-back) or by markup order (the order in which they appear in XAML view).
15	Timeline zoom Set the zoom resolution of the timeline. Zooming in lets you edit an animation with more detail, and zooming out shows more of an overview of what is happening over longer periods of time. If you zoom in but can't set a keyframe at the position in time that you want, verify that the snap resolution is set high enough.
16	Timeline composition area View the timeline, and move keyframes around by dragging them or using their shortcut menus.

Tour of the Properties panel

Use this panel to view and modify the properties of an object. You can also set them directly on the artboard. If you do, the property changes will be reflected in the **Properties** panel.



Categories Expand and collapse categories of properties. Click **Expand**  and **Collapse**  to show or hide category details.

1	Name and Type View the icon, name and type of the selected object.
2	Arrange by Arrange properties alphabetically by name, source, or category.
3	Brush properties Set the visual properties for brushes such as Fill brush, Stroke brush, and Foreground brush.
4	Color editor Use for solid color and gradient brushes.
5	Color picker Select a color.
6	Color chips View the initial color, current color, and last color

7	Eyedroppers Use the color of any element on your screen. The Color eyedropper is available when the Solid color brush is selected. The Gradient eyedropper is available when the Gradient brush is selected.
8	Properties and Events Set properties or choose events for a selected element.
9	Search box Search for properties. Filter the properties that are displayed by typing in the Search box.
10	Brush editor tabs Use to select a brush editor. You can choose No brush , Solid Color brush , Gradient brush , Tile brush , or Brush resource .
11	Color resources Apply the exact same color to different properties. The Color Resources tab includes Local Resources and System Resources .
12	RGB color space Modify the color by adjusting the values for the R , G , or B (red, green, blue) number editors.
13	Alpha channel Modify the Alpha value by using the number editor next to A .
14	Convert color to resource Convert the selected color to a color resource. Color resources are available when you click the Color resources tab.
15	Hex value View the hexadecimal value of the color displayed.
16	Gradient slider Appears only if a gradient brush is selected.
17	Show advanced properties View categories of properties that are less commonly used.

Watch a short video:  [Properties panel](#).

See also

- [Insert controls and modify their behavior](#)
- [Animate objects](#)
- [Draw shapes and paths](#)
- [Designing XAML in Visual Studio and Blend for Visual Studio](#)

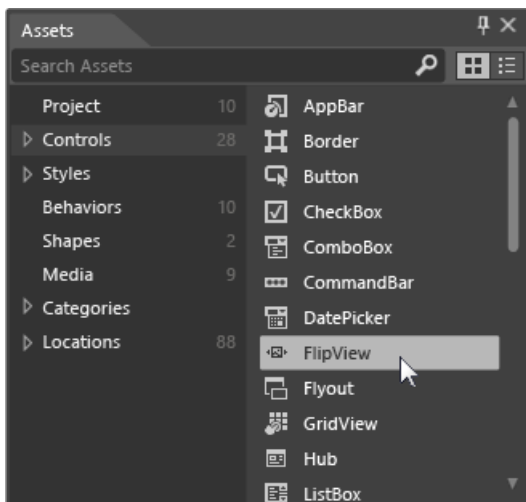
Insert controls and modify their behavior in XAML Designer

2/8/2019 • 2 minutes to read • [Edit Online](#)

Controls enable users to interact with your app. You can use them to collect information and to perform actions such as animate an object or query a data source.

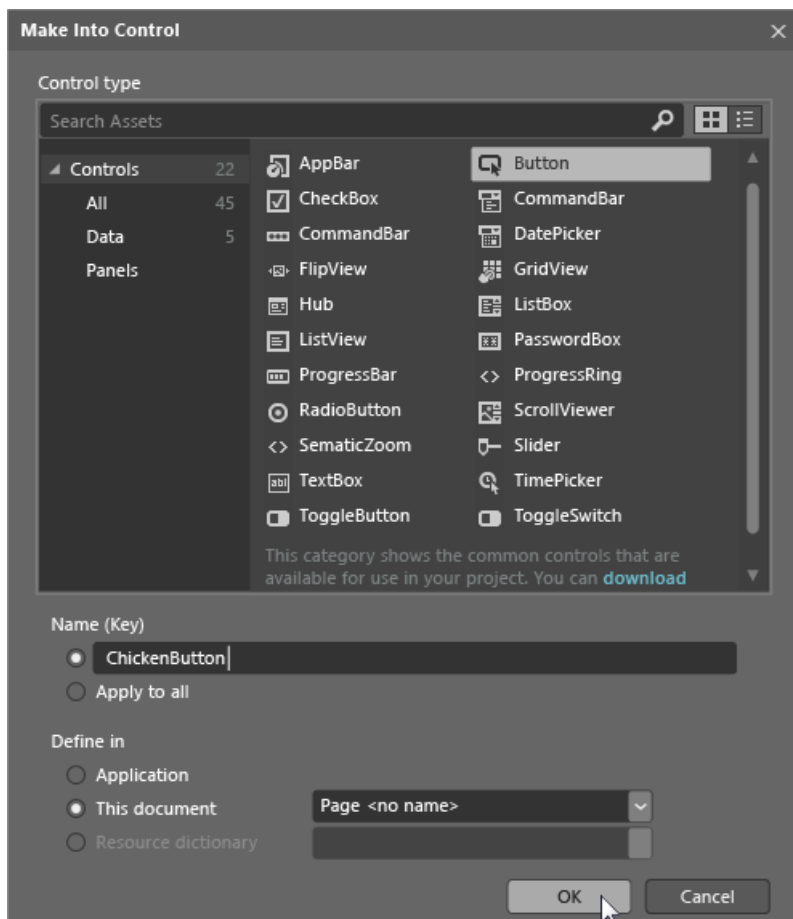
Add controls to the artboard

You can drag controls from the **Assets** panel onto the **artboard**, and then modify them in the **Properties** window.



Make a control out of an image, shape, or path

You can make any object into a control.



For example, imagine a picture of a television in the center of a page. You could make controls out of small images that look like television buttons. Then, users could click those buttons to change the channel.

This is possible because the buttons are now controls. With controls, you can respond to user interactions; in this case, when the user clicks a button.

To make a control, select an object. Then, on the **Tools** menu, click **Make Control**.

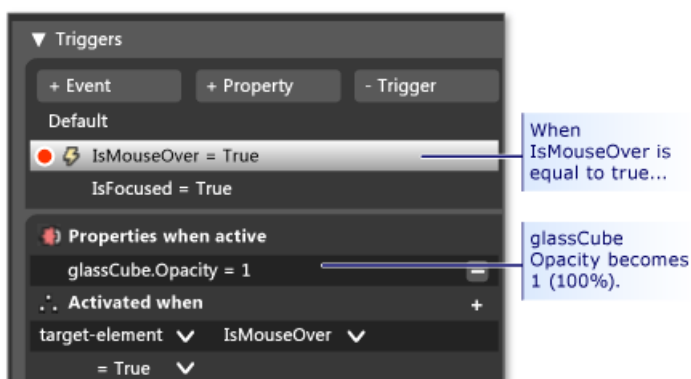
Make controls do things

Controls can perform actions when users interact with them. For example, they can start an animation, update a data source, or play a video.

Use *triggers*, *behaviors*, and *events* to make controls do things.

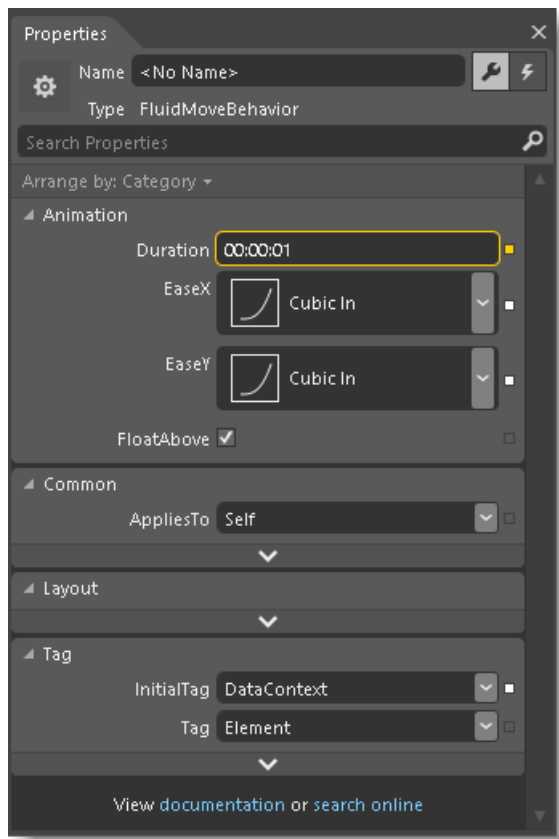
Triggers


A *trigger* changes a property or performs a task in response to an event or a change in another property. For example, you can change the color of a button when users hover over it.



Behaviors

A *behavior* is a reusable package of code. It can do a bit more than change properties. It can perform actions such as query a data service. Blend comes with a small collection of behaviors, but you can add more. Drag a behavior onto any object in your artboard, and then customize the behavior by setting properties.



Watch a video:  [Blend tips: Intro to using behaviors Part 1.](#)

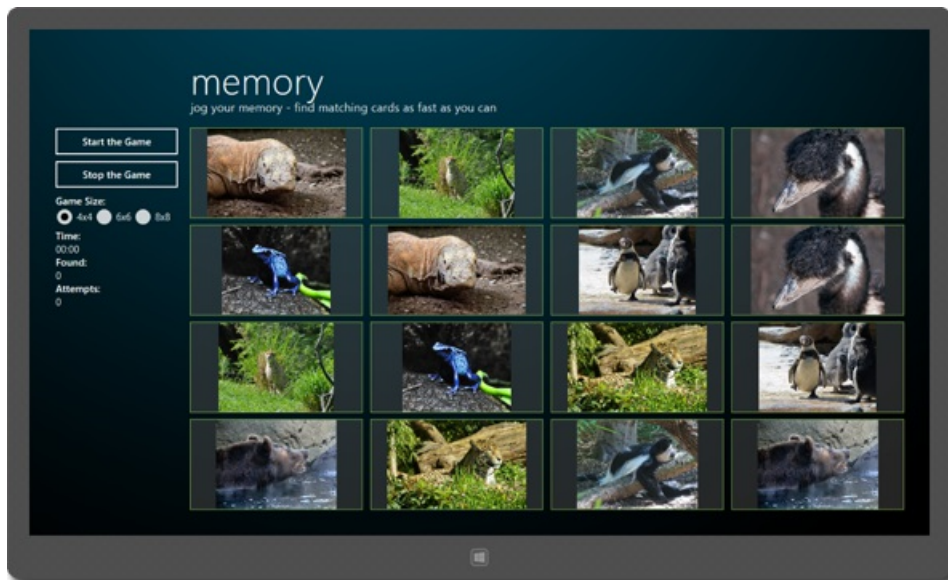
Events

For ultimate flexibility, handle an *event*. You'll have to write some code.

Insert images, videos, and audio clips in XAML Designer

2/8/2019 • 2 minutes to read • [Edit Online](#)

Images, videos, and audio clips add visual appeal to your app.



To use an image, video, or audio clip, add it to your project, and then drag it from the **Assets** panel onto the **artboard**.

These videos can help you insert images, videos, and audio clips into your app.

TASK	WATCH A SHORT VIDEO
Import an Adobe FXG File	FXG Import Preview in Blend
Import an Adobe Illustrator file	Import an Adobe Illustrator (ai) file into Blend
Import an Adobe Photoshop file	Import a Photoshop file into Blend
Insert audio clips	Add audio clips

See also

- [Creating a UI by using Blend for Visual Studio](#)

Draw shapes and paths

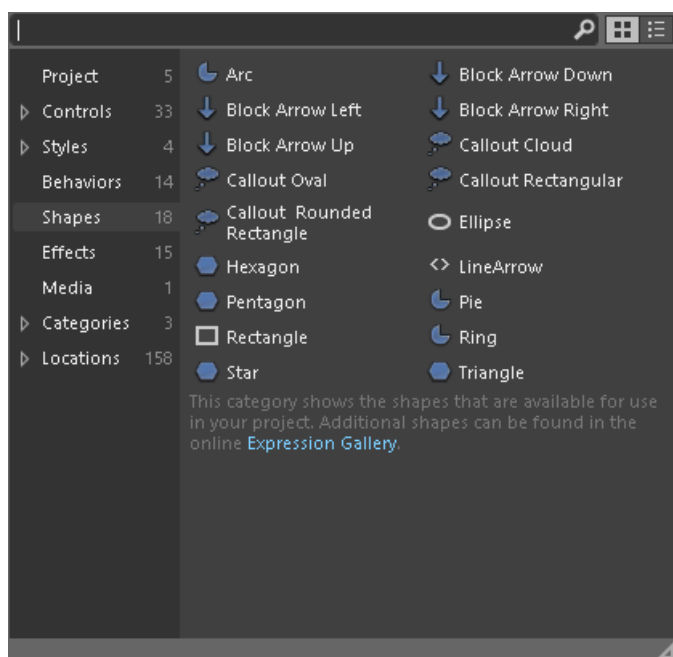
2/8/2019 • 3 minutes to read • [Edit Online](#)

In XAML Designer, a *shape* is exactly what you'd expect. For example: a rectangle, circle, or ellipse. A *path* is a more flexible version of a shape. You can do things like reshape them or combine them together to form new shapes.

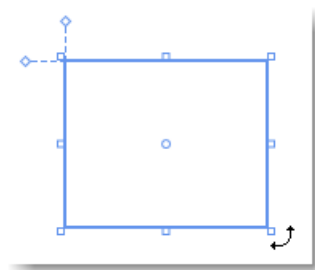
Shapes and paths use vector graphics so they scale well to high resolution displays. If you want to learn more about vector graphics, see [What are Vector Graphics](#) or [vector graphics](#).

Draw a shape

You can find shapes in the **Assets** panel.



Drag any shape that you want to the artboard. Then, you can use handles on the shape to scale, rotate, move, or skew the shape.



Draw a path

A path is a series of connected lines and curves. Use a path to create interesting shapes that are not available in the **Assets** panel.

You can draw a path by using a line, pen or pencil. You can find these tools in the **Tools** panel.



Draw a straight line

Use the **Pen** tool , or the **Line** tool .

Using the Pen tool

On the artboard, click once to define the start point, and then click again to define the end of the line.

Using the Line tool

On the artboard, drag from where you want the line to start, and then release at the point where you want the line to end.

Draw a curve

Use the **Pen** tool .

On the artboard, click once to define the start point of a line, and then click and drag your pointer to create the desired curve.

If you want to close the path, click the first point on the line.

Change the shape of a curve

Use the **Direct selection** tool .

Click the shape, and then drag any point on the shape to change curve shapes.

Draw a free-form path

Use the **Pencil** tool .

On the artboard, draw a free-form path just as you would by using a real pencil.



Remove part of a path

Use the **Direct selection** tool .

Select the path that contains the segment you want to delete, and then click the **Delete** button.



Remove a point in a path

Use the **Selection** tool , and the **Pen** tool .

Use the **Selection** tool  to select the path. Then, use the **Pen** tool  to click the point that you want to remove.

Add a point to a path

Use the **Selection** tool , and the **Pen** tool .

Use the **Selection** tool  to select the path. Use the **Pen** tool  to click anywhere on the path where you want to add the point.

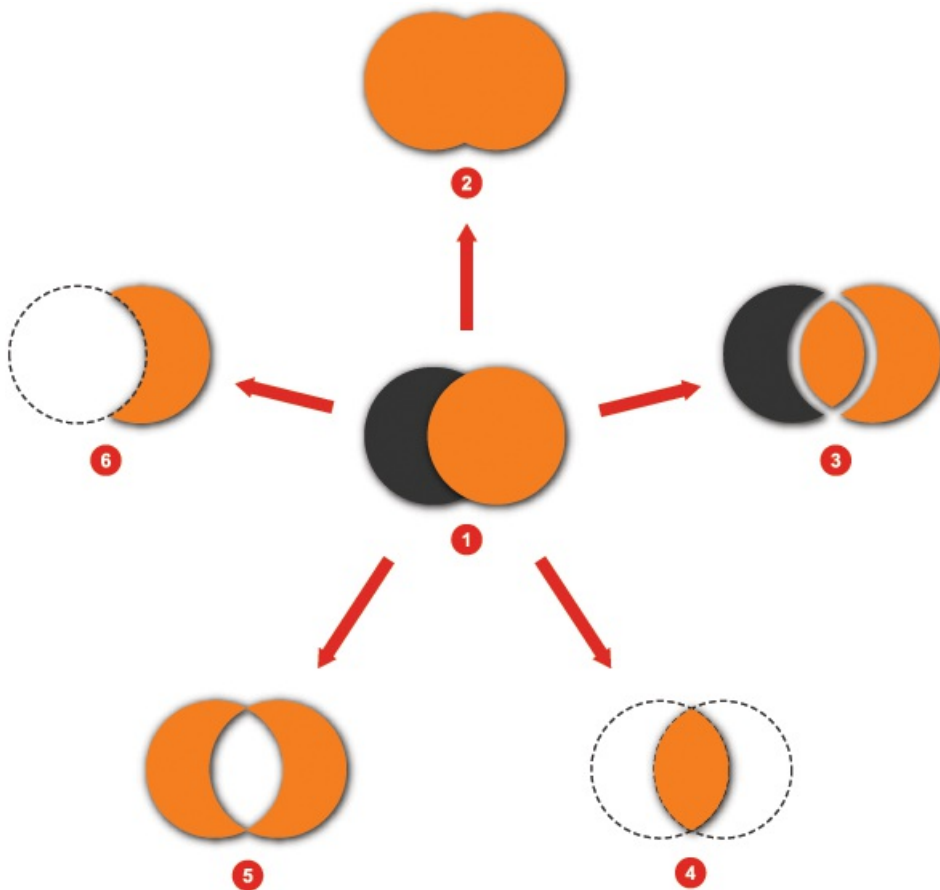
Convert a shape to a path

To modify a shape in the same ways that you modify a path, convert the shape to a path.

Watch a short video:  [Working with paths: Convert a shape to a path.](#)

Combine paths

You can combine paths and shapes into a single path.



1	Two shapes before combining	4	Intersect
2	Unite	5	Exclude Overlap
3	Divide	6	Subtract

Watch a short video: [Working with paths: Combine paths.](#)

Create a compound path

When you create a compound path, any intersecting parts of the paths are subtracted from the result, and the resulting path takes on the visual properties of the bottommost path.

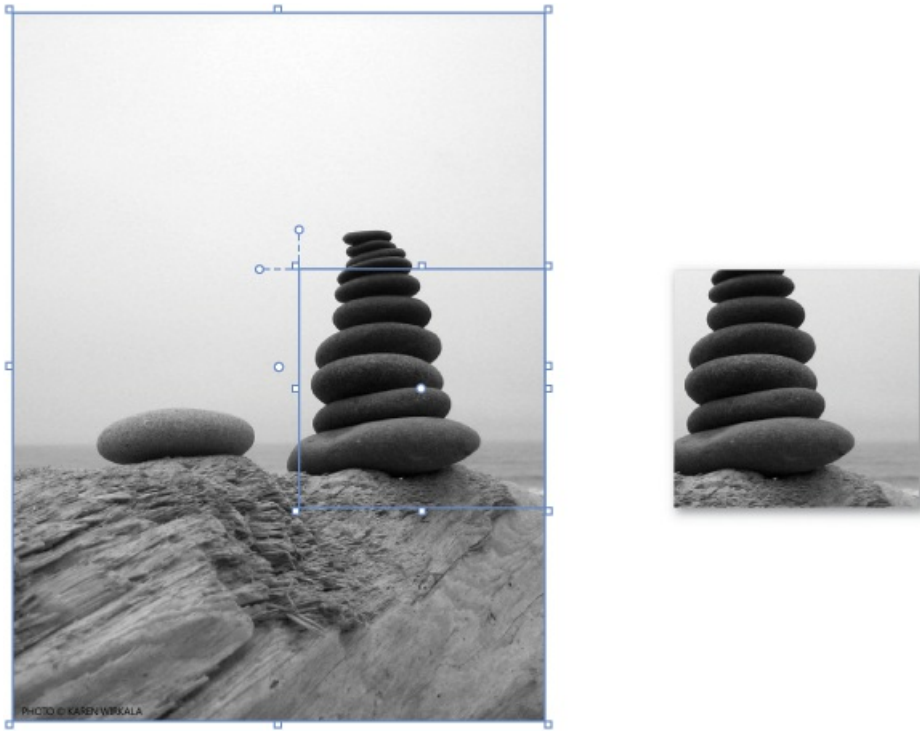
You can break apart a compound path any time after you create it.



Watch a short video:  [Working with paths: Create a compound path.](#)

Create a clipping path

A clipping path is a path or shape that is applied to another object, hiding the parts of the masked object that fall outside the clipping path.



Watch a short video:  [Working with paths: Create a clipping path.](#)

See also

- [Creating a UI by using Blend for Visual Studio](#)

Modify the style of objects in Blend

2/8/2019 • 3 minutes to read • [Edit Online](#)

The easiest way to customize an object is to set properties in the **Properties** pane.

If you want re-use settings or groups of settings, create a re-usable resource. This could be a *style*, *template*, or something simple like a custom color. You can also make a control appear differently based on its state. For example, a button turns green when the user clicks it.

Brushes: Modify the appearance of an object

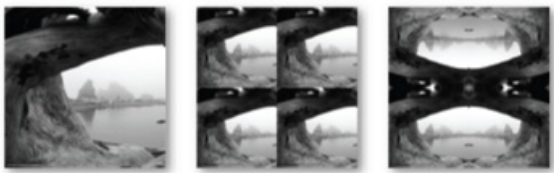
Apply a brush to an object if you want to change its appearance.

Paint a repeating image or pattern on an object

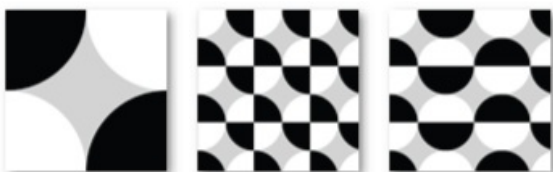
Paint a repeating image or pattern on an object by using a *tile brush*.

To create a tile brush, you begin by creating an *image brush*, *drawing brush*, or *visual brush* resource.

Create an image brush by using an image. The following illustrations show the image brush, the image brush tiled, and the image brush flipped.



Create a drawing brush by using a vector drawing such as a path or shape. The following illustrations show the drawing brush, the drawing brush tiled, and the drawing brush flipped.



Create a visual brush from a control such as a button. The following illustrations show the visual brush, and the visual brush tiled.



Styles and templates: Create a consistent look and feel across controls

You can design the appearance and behavior of a control one time and apply that design to other controls so that you don't have to maintain them individually.

Should you use a style?: If you just want to set default properties (such as the color of a button), use a *style*. You can modify a control even after you've applied a style to it.

Should you use a template?: If you want to change the structure of a control, use a *template*. Imagine converting

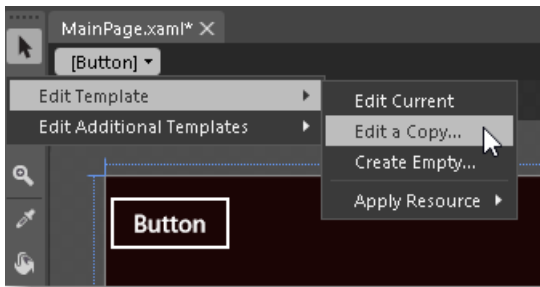
a graphic or logo to a button. You can't modify a control after you've applied a template to it.

Create a template or style

There're two ways to create a template. You can convert any object on your artboard to a control or you can base your template on an existing control.

To convert any object to a control template, select the object, and then on the **Tools** menu, choose **Make Into Control**.

If you want to base your template on an existing control, select an object on the artboard. Then, at the top of the artboard, choose the breadcrumb button, choose **Edit Template**, and then choose **Edit a Copy** or **Create Empty**.

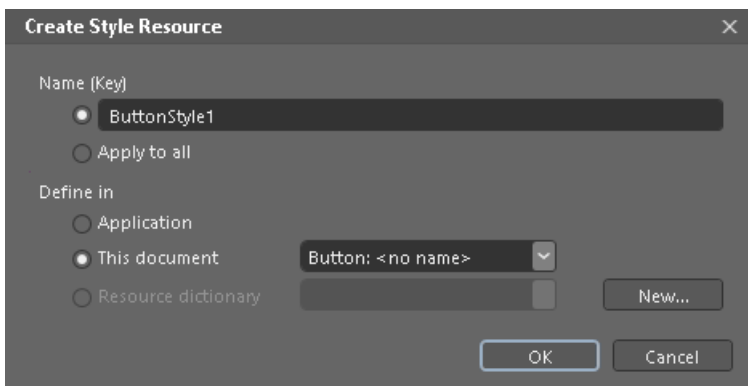


To create a style, select the object, and then in the **Object** menu, choose **Edit Style**, and then choose **Edit a Copy** or **Create Empty**.


- Choose **Edit a Copy** to start with the default style or template of the control.
- Choose **Create Empty** to start from scratch.

The **Edit Current** option appears only if you edit a style or template that you've already created. It won't appear for a control that is still using a default system template.

In the **Create Style Resource** dialog box, you can either name the style or template so that you can use it later, or you can apply the style or template to all controls of that type.

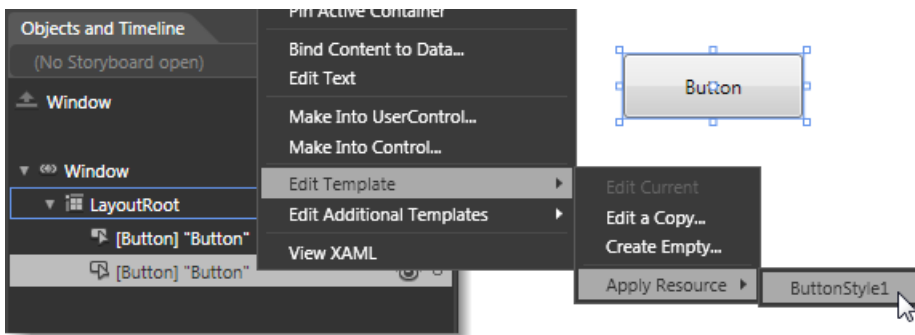


NOTE

You can't create styles or templates for every type of control. If a control doesn't support them, the breadcrumb button won't appear above the artboard. To return to the editing scope of your main document, click **Return scope to** .

Apply a style or template to a control

Right-click an object in the **Objects and Timeline** panel, choose **Edit Template**, and then choose **Apply Resource**.

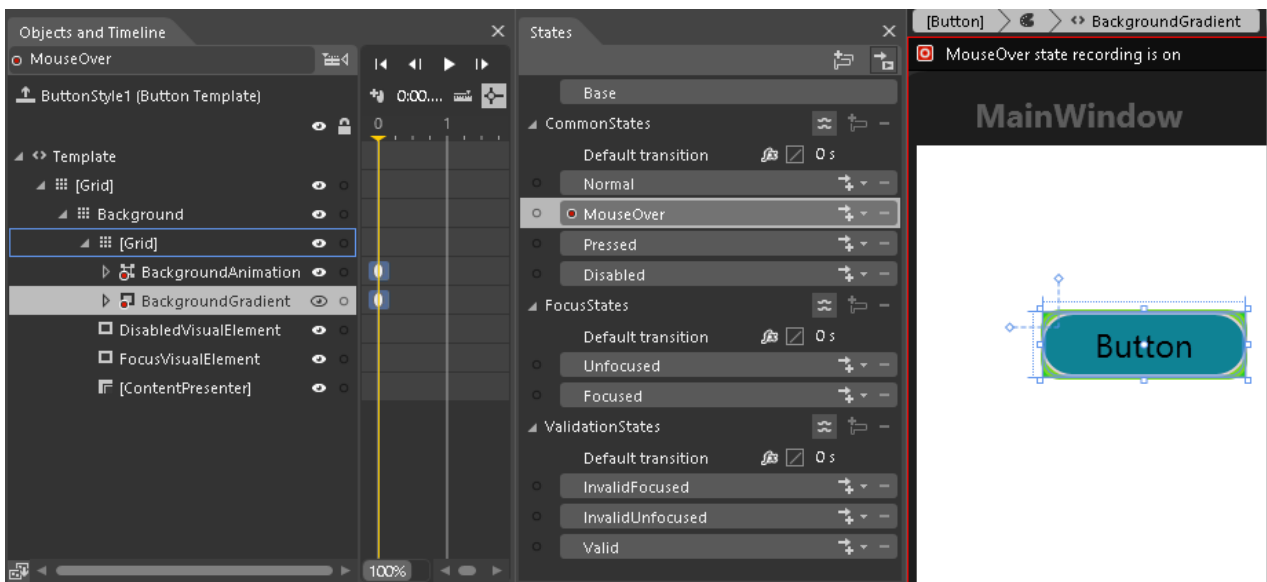


Restore the default style or template of a control

Select the control, and in the [Properties](#) panel, locate the **Style** or **Template** property. Choose **Advanced options**, and then click **Reset** on the shortcut menu.

Visual states: Change the appearance of a control based on its state

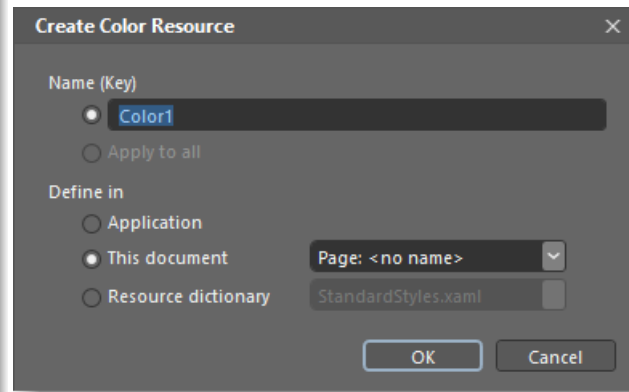
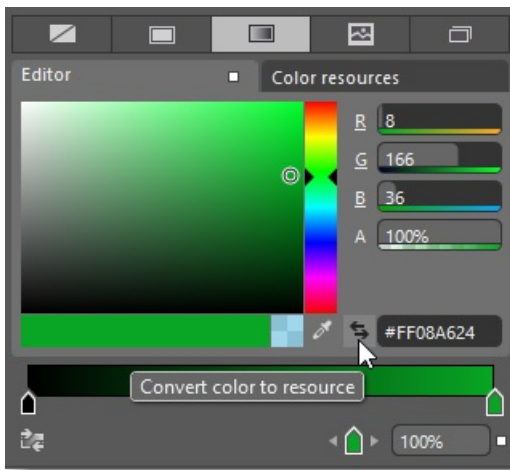
Controls can have different visual appearances based on user interactions. For example, you can make a button turn green when a user clicks it or you can run an animation. You shorten or lengthen the time between visual states by using transitions.



Watch a short video: [Manage the state of your WPF controls.](#)

Resources: Create colors, styles, and templates and reuse them later

You can convert just about anything in your project to a resource. A resource is just an object that you can reuse in different places in your application. For example, you can create a color one time, make it a resource, and then use that color on several objects. To change the color of all of those objects, just change the color resource.



See also

- [Creating a UI by using Blend for Visual Studio](#)

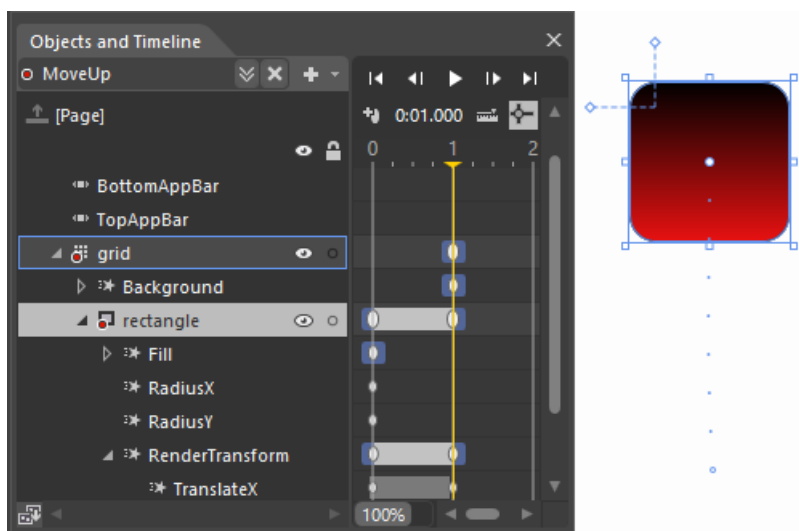
Animate objects in XAML Designer

2/8/2019 • 2 minutes to read • [Edit Online](#)

You can create short animations that move objects, or fade them in and out.

To get started, create a *storyboard*. A storyboard contains one or more *timelines*. Set *keyframes* on a timeline to mark property changes. Then, when you run the animation, Blend interpolates the property changes over the designated period of time. The result is a smooth transition. You can animate any property that belongs to an object, even nonvisual properties.

The following illustration shows a storyboard named **MoveUp**. The timeline contains keyframes that mark the X and Y position of a rectangle. When this animation runs, the rectangle moves from one position to another smoothly.



See also

- [Create a UI by using Blend for Visual Studio](#)

Display data in Blend

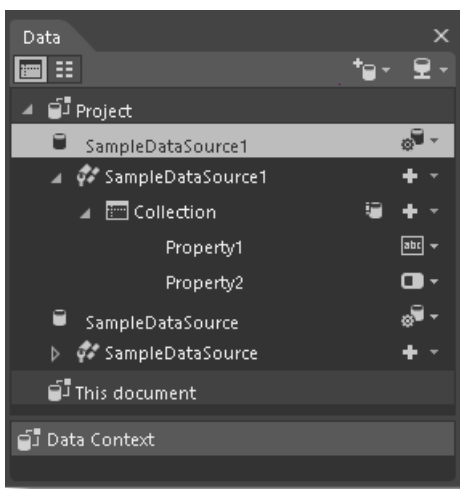
2/8/2019 • 2 minutes to read • [Edit Online](#)


You can view sample data in your designer as you customize the layout of your pages. You can generate sample data from scratch or by using an existing class. You can also connect to *Live data* that appears in your app when you run it.

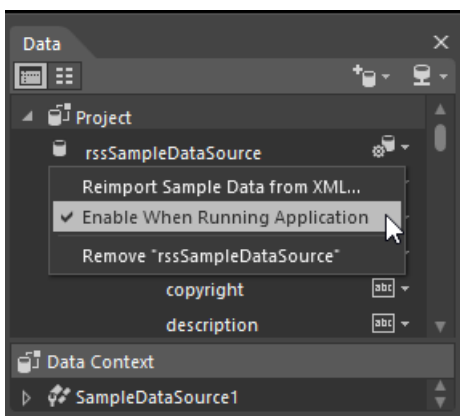
Generate sample data

To generate sample data, open a XAML document. In the **Data** panel, choose the **Create sample data**  button, and then choose **New Sample Data**.

Define the structure of your data in the **Data** panel, and then bind it to UI elements on any page.



If you want your sample data to appear in your pages when you run the app, choose **Data source options** , and then choose **Enable When Running Application**.




Watch a short video:  [Create sample data from scratch.](#)

Watch a short video:  [Mixing up some data binding with Blend.](#)

Generate sample data from a class

If you've already created classes that describe the structure of your data, you can generate sample data from them.

To generate sample data from a class, open a XAML document, and then in the **Data** panel, click the **Create sample data**  button, and then click **Create Sample Data from Class**.

Watch a short video:  [Create sample data from a class.](#)

Watch a short video:  [Mix up some data binding with Blend.](#)

Show live data in a WPF application

Watch a short video:  [Create an XML data source.](#)

Show live data in a store or phone app

See [Work with data and files \(XAML\)](#).

See also

- [Create a UI by using Blend for Visual Studio](#)

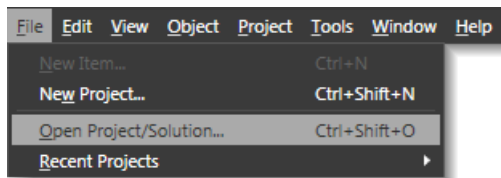
Keyboard shortcuts and modifier keys in Blend

2/8/2019 • 2 minutes to read • [Edit Online](#)

Keyboard shortcuts can speed up your work by reducing an action that would require multiple mouse-button clicks to a single keyboard shortcut. Keyboard shortcuts in Blend for Visual Studio come in the following two categories:

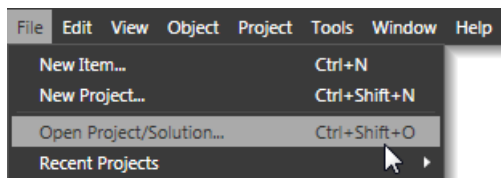
- **Access keys** You can use access keys to access a menu command or an area in a dialog box by pressing a specific key on the keyboard. Access keys are identified by underscores that appear in the currently selected command or dialog box.

To use access keys, first press **Alt** or **F10** to make the underscores appear, and then press the corresponding letter of the specific menu or dialog box item. Alternatively, you can navigate through a menu or dialog box by using the **Tab** key or the arrow keys. For example, if you press **Alt** in Blend, an underline appears under the letter **F** in the **File** menu to identify it as an access key. To open a project, you could press and hold **Alt**, press **F**, and then press **O**.



- **Shortcut keys** You can use shortcut keys to perform an action (such as selecting a menu command or modifying the behavior of a tool) by pressing a keyboard shortcut.

Most keyboard shortcuts are easy to identify in the user interface of Blend; they are displayed to the right of menu commands. For example, on the **File** menu, the **Open Project** menu command includes the keyboard shortcut **Ctrl+Shift+O**. To see the shortcut keys for a tool in the **Tools** panel, hover the pointer over the tool.



For more information about accessibility and features, see [Accessibility at Microsoft](#).

Modifier keys

Some keyboard shortcuts do not have associated menu items, which means that you can't use the Blend user interface to discover them. The following topics list shortcuts that modify the behavior of tools, or that modify an action, such as resizing an object:

- [Artboard modifier keys](#)
- [Pen tool modifier keys](#)
- [Direct Selection tool modifier keys](#)

Keyboard shortcuts in Blend

2/8/2019 • 3 minutes to read • [Edit Online](#)

Project shortcuts

TO DO THIS	DO THIS
Create a new project	Ctrl+Shift+N
Open a project or solution (not a site)	Ctrl+Shift+O
Close a solution	Ctrl+Shift+C
Save a copy of the solution or site	Ctrl+Shift+P
Add an existing item to the project	Ctrl+I
Add a reference to a DLL (WPF)	Alt+Shift+R
Build the project	Ctrl+Shift+B
Test the project or site	F5

Document shortcuts

TO DO THIS	DO THIS
Switch between open documents	Ctrl+Tab
Save the active document	Ctrl+S
Save all documents	Ctrl+Shift+S
Close the active document	Ctrl+W
Close all open documents	Ctrl+Shift+W
Undo the last action	Ctrl+Z
Redo the last action that was undone	Ctrl+Y or Ctrl+Shift+Z
Create a design-time annotation	Ctrl+Shift+T
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V

TO DO THIS	DO THIS
Delete	Delete
Find text (XAML view or JavaScript editor only)	Ctrl+ F
Find the next occurrence of text (XAML view or JavaScript editor only)	F3 or Ctrl+H

Object shortcuts

TO DO THIS	DO THIS
Create a new item	Ctrl+ N
Duplicate an object	Hold down Alt and drag the object
Reparent an object	Drag the object over a layout panel and press Alt
Edit the text in a control	F2 (Esc to exit)
Edit a control (WPF)	Ctrl+ E
Make the selected objects the same width	Ctrl+ Shift+ 1
Make the selected objects the same height	Ctrl+ Shift+ 2
Make the selected objects the same size	Ctrl+ Shift+ 9
Flip the selected object horizontally	Ctrl+ Shift+ 3
Flip the selected object vertically	Ctrl+ Shift+ 4
Select multiple objects	Hold down Ctrl
Select multiple adjacent objects	Hold down Shift
Fit the selection to the screen size	Ctrl+ 9
Pin active container	Ctrl+ Shift+ D
Nudge selected objects	Arrow keys
Auto size width	Ctrl+ Shift+ 5
Auto size height	Ctrl+ Shift+ 6
Group objects into a layout panel	Ctrl+ G
Ungroup objects	Ctrl+ Shift+ G
Bring the selected object to the front	Ctrl+ Shift+]

TO DO THIS	DO THIS
Bring forward	Ctrl+]
Send the selected object to the back	Ctrl+Shift+[
Send backward	Ctrl+[
Make a user control from the selected objects (WPF)	F8
Constrain proportions of objects	Hold down Shift while dragging the object
Rotate an object in 15-degree increments	Hold down Shift while rotating the object
Make a clipping path	Ctrl+7
Release a clipping path	Ctrl+Shift+7
Make a compound path	Ctrl+8
Release a compound path	Ctrl+Shift+8
Lock selection	Ctrl+L
Unlock all objects	Ctrl+Shift+L
Show selection	Ctrl+T
Hide selection	Ctrl+3
Select all objects	Ctrl+A
Clear selection of all objects	Ctrl+Shift+A

View shortcuts

TO DO THIS	DO THIS
Switch between Design , Code , and Split views	F11
Zoom in on the artboard	Ctrl+Equal Sign (=)
Zoom out on the artboard	Ctrl+Minus Sign (-)
Zoom in or out on the artboard	Rotate the mouse wheel
Move the artboard left or right	Shift and rotate the mouse wheel
Move the artboard up or down	Ctrl and rotate the mouse wheel
Fit the selection to the screen size	Ctrl+0

TO DO THIS	DO THIS
View the artboard at actual size	Ctrl+1
Show or hide handles	F9
Show or hide object boundaries	Ctrl+Shift+H
Switch between Design , XAML , and Split views	F11

Workspace shortcuts

TO DO THIS	DO THIS
Switch between Animation and Design workspaces	Ctrl+F11
Show or hide the Assets panel	Ctrl+Period
Show or hide the Results panel	F12
Show or hide all panels	F4
Reset the active workspace layout	Ctrl+Shift+R
Pan the workspace	Hold down Spacebar
Temporarily use the Selection tool (while a different tool remains selected)	Hold down Ctrl

Artboard modifier keys in Blend

2/8/2019 • 2 minutes to read • [Edit Online](#)

Some keyboard shortcuts do not have associated menu items, which means that you can't use the Blend user interface to discover them. The following table lists shortcuts that modify an action, such as resizing an object.


TO DO THIS ACTION	DO THIS
Temporarily select the Selection tool while a different tool remains selected (this reduces the number of times you have to click something in the Tools panel, going back and forth between the Selection tool and others)	Hold down Ctrl
Nudge selected objects while the Selection tool is selected	Press the arrow keys
Pan the artboard	Hold down Spacebar and drag the artboard
Zoom in and out on the artboard	Rotate the mouse wheel
Zoom in on the artboard	Hold down Ctrl+Spacebar while clicking anywhere on the artboard
Zoom out on the artboard	Hold down Ctrl+Alt+Spacebar while clicking anywhere on the artboard
Move the artboard left and right	Hold down Shift and rotate the mouse wheel
Move the artboard up and down	Hold down Ctrl and rotate the mouse wheel
Constrain proportions of objects being drawn or transformed	Hold down Shift
Rotate an object in 15-degree increments	Hold down Shift
Duplicate an object	Hold down Alt and drag the object
Reparent an object	Drag the object over a layout panel and press Alt before releasing the mouse button
Select multiple objects	Hold down Ctrl while selecting each object
Select multiple adjacent objects	Hold down Shift while selecting the first and last objects
Select by drawing a marquee	Hold down Shift and drag
Select an object underneath another object	Hold down Alt and click once for each layer of objects
Switch between open documents	Press Ctrl+Tab
Open the Assets panel	Press Ctrl+Period











See also




- [Keyboard shortcuts](#)
- [Pen tool modifier keys](#)
- [Direct Selection tool modifier keys](#)

Pen tool modifier keys in Blend

2/8/2019 • 2 minutes to read • [Edit Online](#)

The following table lists shortcuts that you can use to modify a path while you are creating it with the **Pen** tool . You can also use the **Pen** tool to add or remove points on an existing path, or to join two existing paths.

TO DO THIS ACTION	DO THIS	POINTER
Create a point to start a straight line segment	Click to create the new point	 Pen pointer
Create a point to start a curved line segment	Click to create the new point, and then drag to adjust the tangent handles before releasing the mouse button	 Pen pointer
Adjust the last tangent without the smooth constraint, allowing you to make a sharp corner	Click to create the new point, and then press Alt before releasing the mouse button	 Pen adjust pointer
Split the last tangent so that the tangent end points operate independently, allowing you to make a sharp corner	Click to create the new point, and then hold down Alt and drag before releasing the mouse button	 Pen adjust pointer
Move the tangent end point around the new point in 15-degree increments	Click to create the new point, and then hold down Shift+Alt and drag before releasing the mouse button	 Pen adjust pointer
Reduce the tangent at an end point to zero length	Click the end point	 Pen adjust pointer
Add a new point to an existing path	Click the path at the location where you want the new point	 Pen insert pointer
Remove a point from a path	Hover over an existing point and click	 Pen delete pointer
Close a path with a sharp corner	Click the start point	 Pen close pointer
Close a path with a smooth curve at the corner	Click the start point and drag to modify the tangent handle before releasing the mouse button	 Pen close pointer


TO DO THIS ACTION	DO THIS	POINTER
Create a sharp corner when joining two paths	Select two paths, click the Pen tool, click an end point of one of the paths, and then click an end point of the other path	 Pen join pointer
Create a smooth corner when joining two paths	Select two paths, click the Pen tool, click an end point of one of the paths, and then drag an end point of the other path	 Pen join pointer
Create a new path	Hold down Ctrl and click outside the previous path to stop adding points to the previous path, and then click or drag where you want the new path to begin	 Pen start pointer











See also

- [Keyboard shortcuts and modifier keys](#)
- [Artboard modifier keys](#)
- [Direct Selection tool modifier keys](#)
- [Draw shapes and paths](#)

Direct Selection tool modifier keys in Blend

2/8/2019 • 2 minutes to read • [Edit Online](#)

The following table lists shortcuts that you can use to modify the shape of an existing path with the **Direct Selection** tool . To add or remove points on an existing path, or to join two existing paths, use the **Pen** tool.

TO DO THIS ACTION	DO THIS	POINTER
Make tangent handles appear for a point on a path	Click a point on a path	 Move point pointer
Move a point on a path	Drag a point on a path	 Move point pointer
Make tangent handles appear for a segment between two points on a path	Click a segment of a path	 Move segment pointer
Move a segment between two points on a path	Drag a segment of a path	 Move segment pointer
Change the angle of a tangent for a point on a path	Click a point or a segment of a path to make the tangent handles appear, and then drag one of the tangent end points	 Move tangent pointer
Make a point into a sharp corner or reduce the tangent to zero	Hover over a point, hold down Alt , and then click the point	 Convert point pointer
Make any sharp corner smooth (or, if it's already smooth, change the angle of the curve as it passes through the clicked point)	Hover over a point, hold down Alt , and then drag the point	 Convert point pointer
Change a curve segment into a straight line	Hover over a segment in a path, hold down Alt , and then click the segment	 Convert segment pointer
Take a segment and bend it into a curve to pass through the pointer position	Hover over a segment in a path, hold down Alt , and then drag the segment	 Convert segment pointer
Adjust one end of a tangent independently of the other side	Direct-select a point or a segment, hold down Alt , and then drag a tangent end point	 Convert tangent pointer

See also

- [Keyboard shortcuts and modifier keys](#)
- [Artboard modifier keys](#)
- [Pen tool modifier keys](#)
- [Draw shapes and paths](#)

Accessibility products and services (Blend)

2/8/2019 • 3 minutes to read • [Edit Online](#)

Microsoft is committed to making its products and services easier for everyone to use. The following sections provide information about the features, products, and services that make Microsoft Windows more accessible for people with disabilities:

- Accessibility features of Windows
- Documentation in alternative formats
- Customer service for people with hearing impairments
- For more information

NOTE

The information in this section may apply only to users who license Microsoft products in the United States. If you obtained this product outside of the United States, you can use the subsidiary information card that came with your software package or visit the [Microsoft accessibility site](#) for a list of Microsoft support services telephone numbers and addresses. You can contact your subsidiary to find out whether the type of products and services described in this section are available in your area. Information about accessibility is available in other languages, including Japanese and French.

Accessibility features of Windows

The Windows operating system has many built-in accessibility features that are useful for individuals who have difficulty typing or using a mouse, are blind or have low vision, or who are deaf or have hearing impairments. The features are installed during Setup. For more information about these features, see Help in Windows and the [Microsoft accessibility site](#).

Free step-by-step tutorials

Microsoft offers a series of step-by-step tutorials that provide detailed procedures for adjusting the accessibility options and settings on your computer. This information is presented in a side-by-side format so that you can learn how to use the mouse, the keyboard, or a combination of both.

To find step-by-step tutorials for Microsoft products, see the [Microsoft accessibility site](#).

Assistive technology products for Windows

A wide variety of assistive technology products are available to make computers easier to use for people with disabilities. You can search a catalog of assistive technology products that run on Windows at the [Microsoft accessibility site](#).

If you use assistive technology, be sure to contact your assistive technology vendor before you upgrade your software or hardware to check for possible compatibility issues.

Documentation in alternative formats

If you have difficulty reading or handling printed materials, you can obtain the documentation for many Microsoft products in more accessible formats. You can view an index of accessible product documentation on the [Microsoft accessibility site](#).

In addition, you can obtain additional Microsoft publications from Recording for the Blind & Dyslexic, Inc (RFB&D).

RFB&D distributes these documents to registered, eligible members of their distribution service. For information about the availability of Microsoft product documentation and books from Microsoft Press, contact:

Learning Ally

20 Roszel Road

Princeton, NJ 08540

Telephone number from within the United States: (800) 221-4792

Telephone number from outside the United States and Canada: (609) 452-0606

Fax: (609) 987-8116

[Learning ally site](#)

Web addresses can change, so you might be unable to connect to the site mentioned here.

Customer service for people with hearing impairments

If you are deaf or have hearing impairments, complete access to Microsoft product and customer services is available through a text telephone (TTY/TDD) service:

For customer service, contact Microsoft Sales Information Center at (800) 892-5234 between 6:30 AM and 5:30 PM Pacific Time, Monday through Friday, excluding holidays.

For technical assistance in the United States, contact Microsoft Product Support Services at (800) 892-5234 between 6:00 AM and 6:00 PM Pacific Time, Monday through Friday, excluding holidays. In Canada, dial (905) 568-9641 between 8:00 AM and 8:00 PM Eastern Time, Monday through Friday, excluding holidays.

Microsoft Support Services are subject to the prices, terms, and conditions in place at the time the service is used.

For more information

For more information about how accessible technology for computers helps to improve the lives of people with disabilities, see the [Microsoft accessibility site](#).

Get started with WPF

2/8/2019 • 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?

SUBJECT	ARTICLES
I want to jump right in...	Walkthrough: My first WPF desktop application
I want to compare XAML design tools...	Design XAML in Visual Studio and Blend for Visual Studio
New to .NET?	Overview of the .NET framework Application Essentials Get Started with Visual C# and Visual Basic
Tell me more about WPF...	WPF overview XAML overview (WPF) Controls Data binding overview WPF data binding with LINQ to XML
Are you a Windows Forms developer?	Windows Forms controls and equivalent WPF controls Supported scenarios in WPF and Windows Forms interoperation

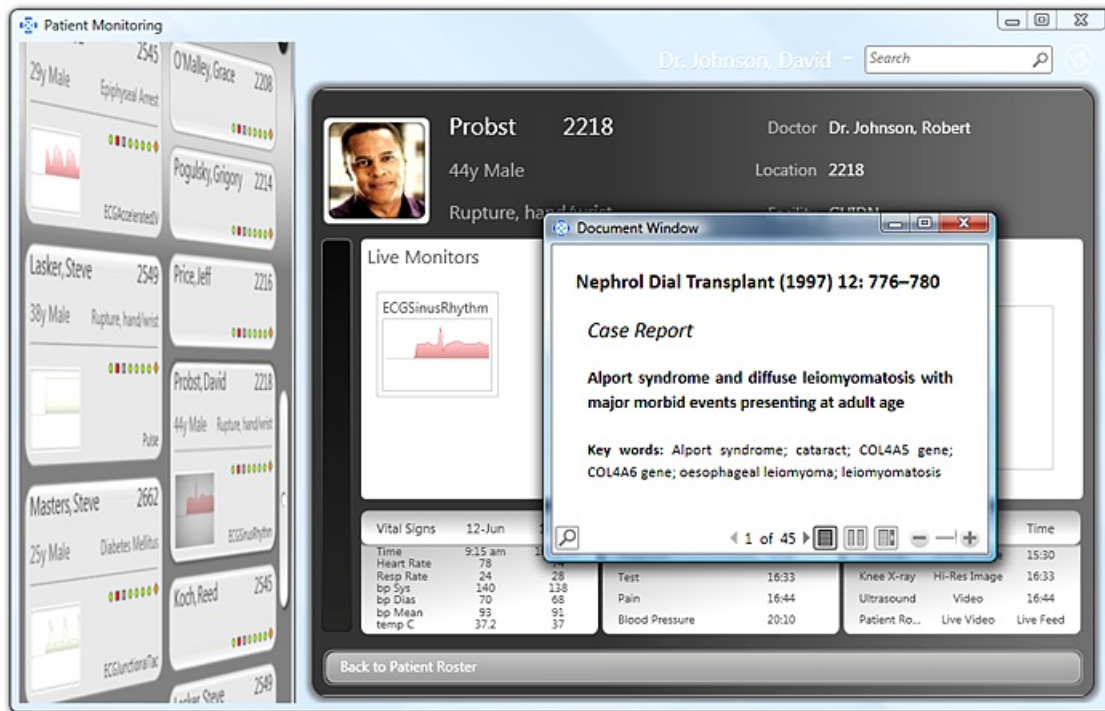
See also

- [Class library \(WPF\)](#)
- [WPF community resources](#)
- [App development overview](#)

WPF overview

2/8/2019 • 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 included in the .NET Framework, so you can build applications that incorporate other elements of the .NET Framework class library.

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

Program with WPF

WPF exists as a subset of .NET Framework types that are for the most part located in the [System.Windows](#) namespace. If you have previously built applications with .NET Framework using managed technologies like ASP.NET and Windows Forms, the fundamental WPF programming experience should be familiar; you instantiate classes, set properties, call methods, and handle events, all using your favorite .NET 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.

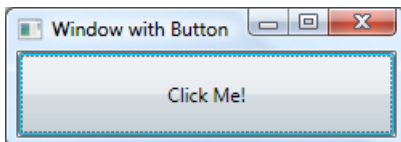
```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="Window with Button"
  Width="250" Height="100">

  <!-- 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

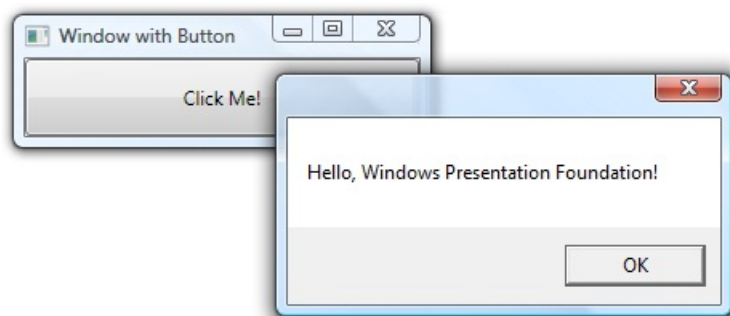
End Namespace

```

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>

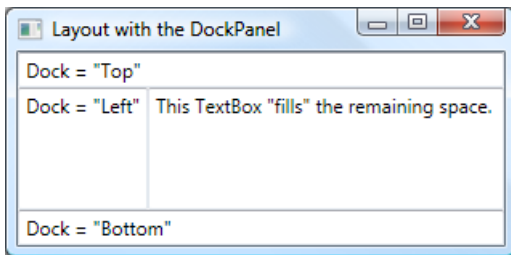
</Window>
```

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.

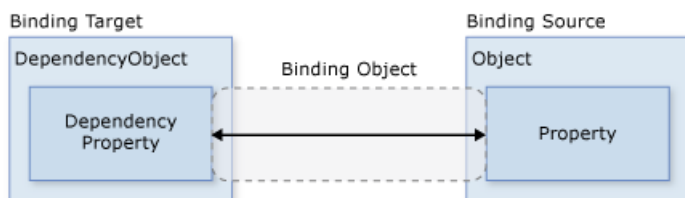


Data binding

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](#).

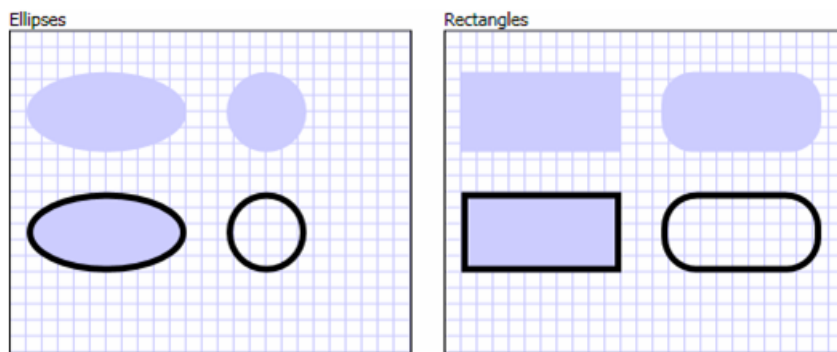
Graphics

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 (sRGB) 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
```



```

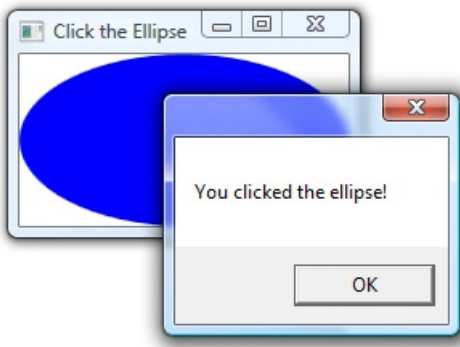
using System.Windows; // Window, MessageBox
using System.Windows.Input; // MouseButtonEventHandler

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.



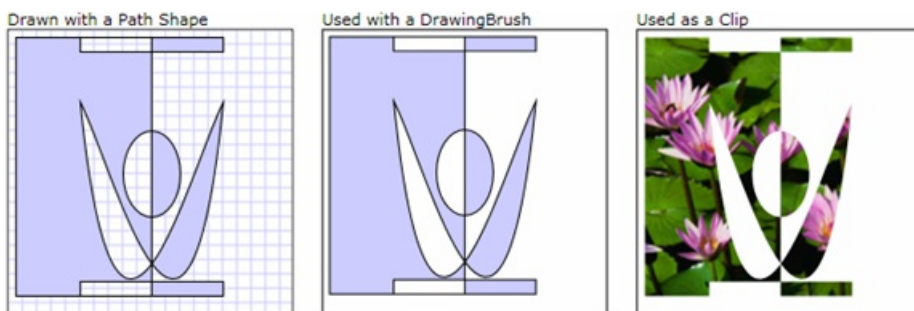
For more information, see [Shapes and basic drawing in WPF overview](#).

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.

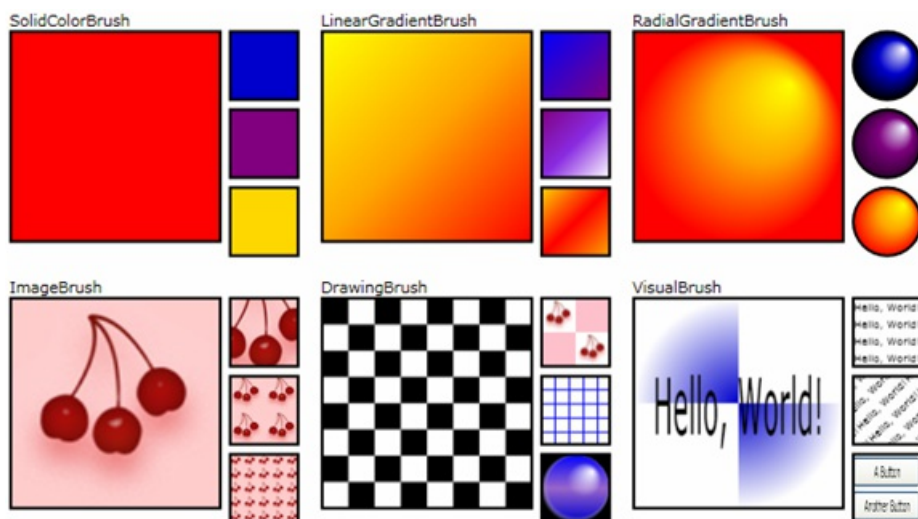
[Geometry](#) objects can be used for clipping, hit-testing, and rendering 2D graphic data.



For more information, see [Geometry overview](#).

2D effects

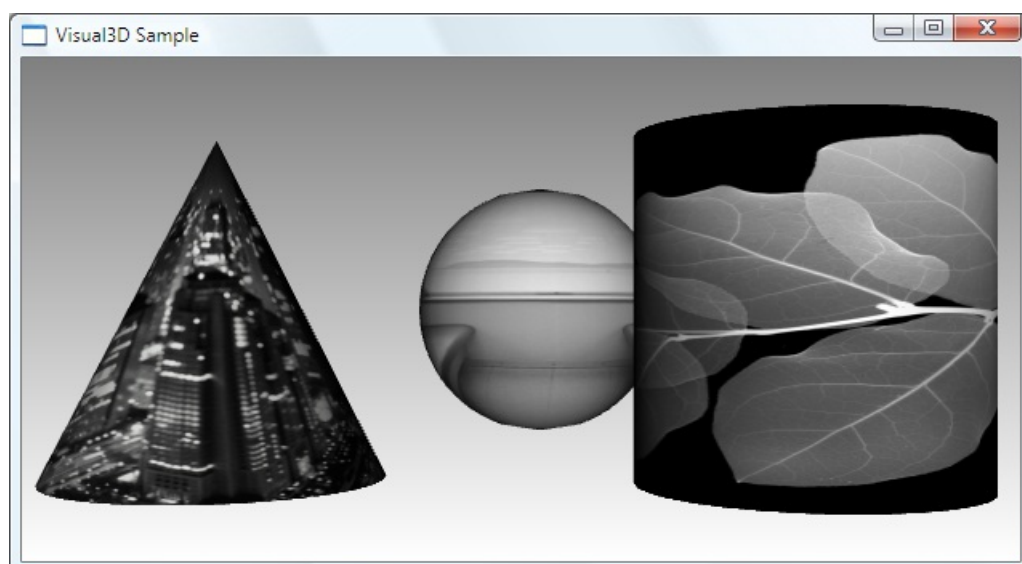
A subset of WPF 2D capabilities includes visual effects, such as gradients, bitmaps, drawings, painting with videos, rotation, scaling, and skewing. These are all achieved with brushes; the following figure shows some examples.



For more information, see [WPF brushes overview](#).

3D rendering

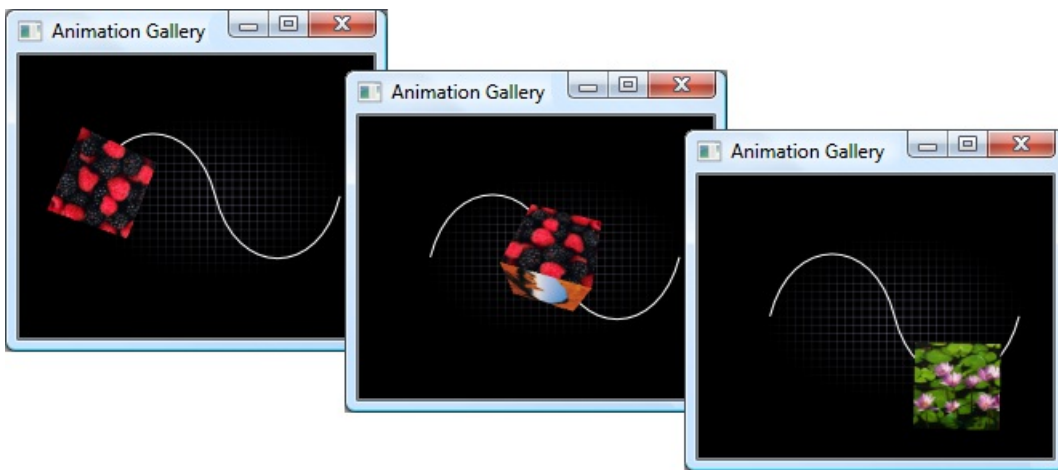
WPF also includes 3D rendering capabilities that integrate with 2-d graphics to allow the creation of more exciting and interesting user interfaces. For example, the following figure shows 2D images rendered onto 3D shapes.



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.



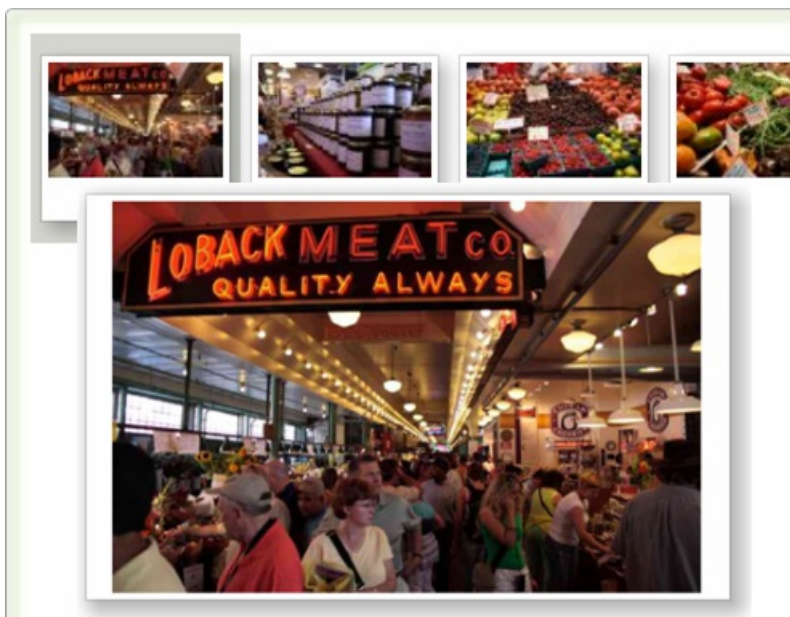
For more information, see [Animation overview](#).

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/wp7.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.

Basic Text Decorations with XAML

The lazy dog ~~The lazy dog~~ The lazy dog The lazy dog

Changing the Color of a Text Decoration with XAML

The lazy dog ~~The lazy dog~~ The lazy dog The lazy dog

Creating Dash Text Decorations with XAML

The lazy dog ~~The lazy dog~~ The lazy dog The lazy dog

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

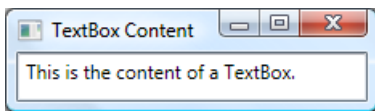
The main purpose of a majority of the WPF controls is to display content. In WPF, the type and number of items that can constitute the content of a control is referred to as the control's *content model*. Some controls can contain a single item and type of content; for example, the content of a [TextBox](#) 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.



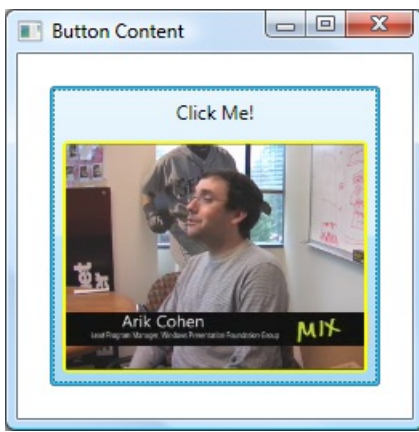
Other controls, however, can contain multiple items of different types of content; the content of a [Button](#), specified by the [Content](#) property, can contain a variety of items including layout controls, text, images, and shapes. The following example shows a [Button](#) with content that includes 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 [Styling and templating](#).

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.

For example, 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

```

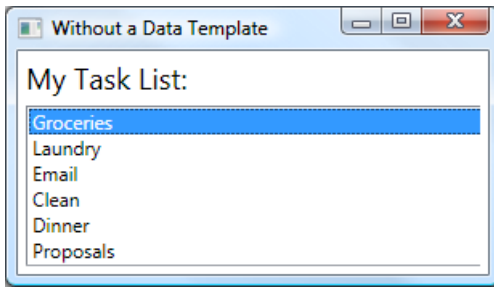
In this example, the default button user interface has been replaced with an [Ellipse](#) that has a dark blue border and is filled using a [RadialGradientBrush](#). The [ContentPresenter](#) control displays the content of the [Button](#), "Click Me!" When the [Button](#) is clicked, the [Click](#) 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.

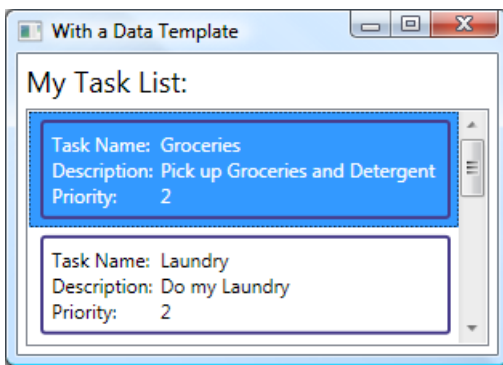
```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SDKSample.DataTemplateWindow"
  Title="With a Data Template">
  <Window.Resources>
    <!-- Data Template (applied to each bound task item in the task collection) -->
    <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 templing overview](#).

Styles

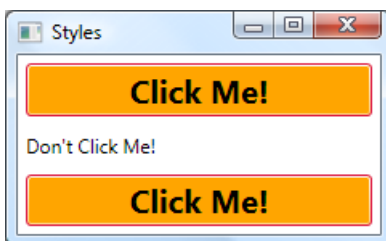
Styles enable developers and designers to standardize on a particular appearance for their product. WPF provides a strong style model, the foundation of which is the [Style](#) element. The following example creates a style that sets the background color for every [Button](#) on a window to Orange.

```
<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 [Styling and templating](#).

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

```

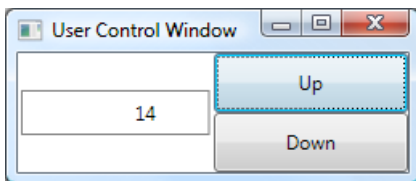
The next example illustrates the XAML that is required to incorporate the user control into a [Window](#).

```
<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](#)

WPF Data Binding with LINQ to XML Overview

2/8/2019 • 4 minutes to read • [Edit Online](#)

This topic introduces the dynamic data binding features in the [System.Xml.Linq](#) namespace. These features can be used as a data source for user interface (UI) elements in Windows Presentation Foundation (WPF) apps. This scenario relies upon special *dynamic properties* of [System.Xml.Linq.XAttribute](#) and [System.Xml.Linq.XElement](#).

XAML and LINQ to XML

The Extensible Application Markup Language (XAML) is an XML dialect created by Microsoft to support .NET Framework 3.0 technologies. It is used in WPF to represent user interface elements and related features, such as events and data binding. In Windows Workflow Foundation, XAML is used to represent program structure, such as program control (*workflows*). XAML enables the declarative aspects of a technology to be separated from the related procedural code that defines the more individualized behavior of a program.

There are two broad ways that XAML and LINQ to XML can interact:

- Because XAML files are well-formed XML, they can be queried and manipulated through XML technologies such as LINQ to XML.
- Because LINQ to XML queries represent a source of data, these queries can be used as a data source for data binding for WPF UI elements.

This documentation describes the second scenario.

Data Binding in the Windows Presentation Foundation

WPF data binding enables a UI element to associate one of its properties with a data source. A simple example of this is a [Label](#) whose text presents the value of a public property in a user-defined object. WPF data binding relies on the following components:

COMPONENT	DESCRIPTION
Binding target	The UI element to be associated with the data source. Visual elements in WPF are derived from the UIElement class.
Target property	The <i>dependency property</i> of the binding target that reflects the value of the data-binding source. Dependency properties are directly supported by the DependencyObject class, which UIElement derives from.
Binding source	The source object for one or more values that are supplied to the UI element for presentation. WPF automatically supports the following types as binding sources: CLR objects, ADO.NET data objects, XML data (from XPath or LINQ to XML queries), or another DependencyObject .
Source path	The property of the binding source that resolves to the value or set of values that is to be bound.

A dependency property is a concept specific to WPF that represent a dynamically computed property of a UI element. For example, dependency properties often have default values or values that are provided by a parent element. These special properties are backed by instances of the [DependencyProperty](#) class (and not fields as with

standard properties). For more information, see [Dependency Properties Overview](#).

Dynamic Data Binding in WPF

By default, data binding occurs only when the target UI element is initialized. This is called *one-time* binding. For most purposes, this is insufficient; typically, a data-binding solution requires that the changes be dynamically propagated at run time using one of the following:

- *One-way* binding causes the changes to one side to be propagated automatically. Most commonly, changes to the source are reflected in the target, but the reverse can sometimes be useful.
- In *two-way* binding, changes to the source are automatically propagated to the target, and changes to the target are automatically propagated to the source.

For one-way or two-way binding to occur, the source must implement a change notification mechanism, for example by implementing the [INotifyPropertyChanged](#) interface or by using a *PropertyNameChanged* pattern for each property supported.

For more information about data binding in WPF, see [Data Binding \(WPF\)](#).

Dynamic Properties in LINQ to XML Classes

Most LINQ to XML classes do not qualify as proper WPF dynamic data sources. Some of the most useful information is available only through methods, not properties, and properties in these classes do not implement change notifications. To support WPF data binding, LINQ to XML exposes a set of *dynamic properties*.

These dynamic properties are special run-time properties that duplicate the functionality of existing methods and properties in the [XAttribute](#) and [XElement](#) classes. They were added to these classes solely to enable them to act as dynamic data sources for WPF. To meet this need, all these dynamic properties implement change notifications. A detailed reference for these dynamic properties is provided in the next section, [LINQ to XML Dynamic Properties](#).

NOTE

Many of the standard public properties, found in the various classes in the [System.Xml.Linq](#) namespace, can be used for one-time data binding. However, remember that neither the source nor the target will be dynamically updated under this scheme.

Accessing Dynamic Properties

The dynamic properties in the [XAttribute](#) and [XElement](#) classes cannot be accessed like standard properties. For example, in CLR-compliant languages such as C#, they cannot be:

- Accessed directly at compile time. Dynamic properties are invisible to the compiler and to Visual Studio IntelliSense.
- Discovered or accessed at run time using .NET reflection. Even at run time, they are not properties in the basic CLR sense.

In C#, dynamic properties can only be accessed at run time through facilities provided by the [System.ComponentModel](#) namespace.

In contrast, however, in an XML source dynamic properties can be accessed through a straightforward notation in the following form:

```
<object>.<dynamic-property>
```

The dynamic properties for these two classes either resolve to a value that can be used directly, or to an indexer

that must be supplied with an index to obtain the resulting value or collection of values. The latter syntax takes the form:

```
<object>.<dynamic-property>[<index-value>]
```

For more information, see [LINQ to XML Dynamic Properties](#).

To implement WPF dynamic binding, dynamic properties will be used with facilities provided by the [System.Windows.Data](#) namespace, most notably the [Binding](#) class.

See also

- [WPF Data Binding with LINQ to XML](#)
- [LINQ to XML Dynamic Properties](#)
- [XAML in WPF](#)
- [Data Binding \(WPF\)](#)
- [Using Workflow Markup](#)

LINQ to XML dynamic properties

2/8/2019 • 2 minutes to read • [Edit Online](#)

This section provides reference information about the dynamic properties in LINQ to XML. Specifically, these properties are exposed by the [XAttribute](#) and [XElement](#) classes, which are in the [System.Xml.Linq](#) namespace.

As explained in the topic [Overview of WPF data binding with LINQ to XML](#), each of the dynamic properties is equivalent to a standard public property or method in the same class. These standard members should be used for most purposes; dynamic properties are provided specifically for LINQ to XML data binding scenarios. For more information about the standard members of these classes, see the [XAttribute](#) and [XElement](#) reference topics.

With respect to their resolved values, the dynamic properties in this section fall into two categories:

- Simple ones, such as the `Value` properties in the [XAttribute](#) and [XElement](#) classes, that resolve to a single value.
- Indexed values, such as the [Elements](#) and [Descendants](#) properties of [XElement](#), that resolve into an indexer type. For indexer types to be resolved to the desired value or collection, an expanded name parameter must be passed to them.

All the dynamic properties that return an indexed value of type [IEnumerable<T>](#) use deferred execution. For more information about deferred execution, see [Introduction to LINQ queries \(C#\)](#).

Reference

- [System.Xml.Linq](#)
- [System.Xml.Linq.XElement](#)
- [System.Xml.Linq.XAttribute](#)

See also

- [WPF data binding with LINQ to XML](#)
- [WPF data binding with LINQ to XML overview](#)
- [Introduction to LINQ queries \(C#\)](#)

XAttribute class dynamic properties

2/8/2019 • 2 minutes to read • [Edit Online](#)

This section describes the dynamic properties of the [System.Xml.Linq.XAttribute](#).

In this section

TOPIC	DESCRIPTION
Value	Gets or sets the value of the XML attribute.

See also

- [System.Xml.Linq.XAttribute](#)
- [LINQ to XML dynamic properties](#)
- [XElement class dynamic properties](#)

Value (XmlAttribute Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets or sets the value of the XML attribute.

Syntax

```
attrib.Value
```

Property value/return value

A [String](#) containing the value of this attribute.

Exceptions

EXCEPTION TYPE	CONDITION
ArgumentNullException	When setting, the <code>value</code> is <code>null</code> .

Remarks

This property is equivalent to the [Value](#) property of the [System.Xml.Linq.XmlAttribute](#) class, but this dynamic property also supports change notifications.

See also

- [System.Xml.Linq.XmlAttribute.Value](#)
- [XmlAttribute class dynamic properties](#)
- [Attribute](#)

XElement class dynamic properties

2/8/2019 • 2 minutes to read • [Edit Online](#)

This section describes the dynamic properties of the [System.Xml.Linq.XElement](#) class.

In this section

TOPIC	DESCRIPTION
Attribute	Gets an indexer used to retrieve the attribute that corresponds to a specified expanded name.
Element	Gets an indexer used to retrieve the child element that corresponds to a specified expanded name.
Elements	Gets an indexer used to retrieve the child elements of the current element that match a specified expanded name.
Descendants	Gets an indexer used to retrieve all the descendant elements of the current element that match a specified expanded name.
Value	Gets or sets the content of an element.
Xml	Gets the unformatted XML representation of an element.

See also

- [System.Xml.Linq.XElement](#)
- [LINQ to XML dynamic properties](#)
- [XAttribute class dynamic properties](#)

Attribute (XElement Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets an indexer used to retrieve the attribute instance that corresponds to the specified expanded name.

Syntax

```
elem.Attribute[{namespaceName}attribName]
```

Property Value/Return Value

An indexer of the type `XAttribute Item(String expandedName)`. This indexer takes the expanded name of the specified attribute and returns the corresponding [XAttribute](#), or `null` if there is no attribute with the specified name.

Remarks

This property is equivalent to the [Attribute](#) method of the [System.Xml.Linq.XElement](#) class.

See also

- [System.Xml.Linq.XElement.Attribute](#)
- [XElement Class Dynamic Properties](#)
- [Value](#)

Element (XElement Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets an indexer used to retrieve the child element instance that corresponds to the specified expanded name.

Syntax

```
elem.Element[{namespaceName}localName]
```

Property Value/Return Value

An indexer of the type `XElement Item(String expandedName)`. This indexer takes an expanded name parameter and returns the corresponding [XElement](#), or `null` if there is no element with the specified name.

Remarks

This property is equivalent to [Element](#) method of the [System.Xml.Linq.XContainer](#) class.

See also

- [System.Xml.Linq.XContainer.Element](#)
- [XElement class dynamic properties](#)
- [Elements](#)

Elements (XElement Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets an indexer used to retrieve the child elements of the current element that match the specified expanded name.

Syntax

```
elem.Elements[{namespaceName}localName]
```

Property Value/Return Value

An indexer of the type `IEnumerable<XElement> Item(String expandedName)`. This indexer takes the expanded name of the desired child elements and returns the matching child elements in an `IEnumerable` `< XElement >` collection.

Remarks

This property is equivalent to the `System.Xml.Linq.XContainer.Elements(XName)` method of the `XContainer` class.

The elements in the returned collection are in XML source document order.

This property uses deferred execution.

See also

- [XElement class dynamic properties](#)
- [Element](#)
- [Descendants](#)

Descendants (XElement Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets an indexer used to retrieve all the descendant elements of the current element that match the specified expanded name.

Syntax

```
elem.Descendants[{namespaceName}localName]
```

Property Value/Return Value

An indexer of the type `IEnumerable<XElement> Item(String expandedName)`. This indexer takes the expanded name of the specified descendant elements and returns the matching child elements in an `IEnumerable<XElement>` collection.

Remarks

This property is equivalent to the `System.Xml.Linq.XContainer.Descendants(XName)` method of the `XContainer` class.

The elements in the returned collection are in XML source document order.

This property uses deferred execution.

See also

- [XElement class dynamic properties](#)
- [Elements](#)

Value (XElement Dynamic Property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets or sets the content of the element.

Syntax

```
elem.Value
```

Property value/return value

A [String](#) that represents the concatenated contents of the element.

Remarks

This property is equivalent to the [Value](#) property of the [System.Xml.Linq.XElement](#) class, but this dynamic property also supports change notifications.

See also

- [System.Xml.Linq.XElement.Value](#)
- [XElement class dynamic properties](#)
- [Xml](#)

Xml (XElement dynamic property)

2/8/2019 • 2 minutes to read • [Edit Online](#)

Gets the unformatted XML content of the element.

Syntax

```
elem.Xml
```

Property value/return value

A [String](#) that represents the unformatted XML content of the element.

Remarks

This property is equivalent to the [ToString\(SaveOptions\)](#) method of the [System.Xml.Linq.XNode](#) class, with the `SaveOptions` parameter set to [SaveOptions](#).

See also

- [XElement class dynamic properties](#)
- [Value](#)

WPF data binding using LINQ to XML example

2/8/2019 • 2 minutes to read • [Edit Online](#)

This section provides a Windows Presentation Foundation (WPF) example that binds user interface components to an embedded XML data source. The name of this example (and the Visual Studio project that contains it) is *LinqToXmlDataBinding*.

In this section

TOPIC	DESCRIPTION
How to: Build and run the LinqToXmlDataBinding example	Contains step-by-step instructions on how to create, populate, and build the Visual Studio project for this example.
Walkthrough: LinqToXmlDataBinding example	Contains the primary source files for the project and a description of how LINQ to XML is used for data binding within this code.

See also

- [WPF data binding with LINQ to XML](#)

How to: Build and run the LinqToXmlDataBinding example

2/8/2019 • 2 minutes to read • [Edit Online](#)

This topic shows how to create and build the LinqToXmlDataBinding Visual Studio project, and how to run the resulting LinqToXmlDataBinding Windows Presentation Foundation (WPF) example program.

For more information about Visual Studio, see [Visual Studio IDE overview](#).

Create and populate the project

To create the starting project

1. Start Visual Studio and create a C# WPF application named LinqToXmlDataBinding. The project must use the .NET Framework 3.5 (or later).
2. If not already present, add project references for the following .NET assemblies:
 - System.Data
 - System.Data.DataSetExtensions
 - System.Xml
 - System.Xml.Linq
3. Build the solution by pressing **Ctrl+Shift+B**, then run it by pressing **F5**. The project should compile without errors and run as a generic WPF application.

To add custom code to the project

1. In Solution Explorer, rename the source file **Window1.xaml** to **L2XDBForm.xaml**. The dependent source file **Window1.xaml.cs** should automatically be renamed to **L2XDBForm.xaml.cs**.
2. Replace the source code found in the file **L2XDBForm.xaml** with the code section from the topic [L2DBForm.xaml source code](#). Use the XAML source view to work with this file.
3. Similarly, replace the source in **L2XDBForm.xaml.cs** with the code found in [L2DBForm.xaml.cs source code](#).
4. In the file **App.xaml**, replace all occurrences of the string `Window1.xaml` with `L2XDBForm.xaml`.
5. Build the solution by pressing **Ctrl+Shift+B**.

Run the program

The LinqToXmlDataBinding program enables the user to view and manipulate a list of books that is stored as an embedded XML element.

To run the program and view the book list

- Run LinqToXmlDataBinding by pressing **F5 (Start Debugging)** or **Ctrl+F5 (Start Without Debugging)**.

A program window with the title **WPF Data Binding using LINQ to XML** appears.
- Notice the top section of the UI, which displays the raw **XML** that represents the book list. It is displayed using a WPF [TextBlock](#) control, which does not enable interaction through the mouse or keyboard.

- The second vertical section, labeled **Book List**, displays the books as a plain text ordered list. It uses a [ListBox](#) control that enables selection through the mouse or keyboard.

To add and delete books from the list

- To add a new book to the list, enter values into the **ID** and **Value**[TextBox](#) controls in the last section, **Add New Book**, then click the **Add Book** button. Note that the book is appended to the list in both the book and XML listings. This program does not validate input values.
- To delete an existing book from the list, select it in the **Book List** section, then click the **Remove Selected Book** button. Notice that the book entry has been removed from both the book and the raw XML source listings.

To edit an existing book entry

1. Select the book entry in the second **Book List** section. Its current values should be displayed in the third section, **Edit Selected Book**.
2. Edit the values using the keyboard. As soon as either [TextBox](#) control loses focus, changes are automatically propagated to the XML source and book listings.

See also

- [WPF data binding using LINQ to XML example](#)
- [Walkthrough: LinqToXmlDataBinding example](#)
- [Visual Studio IDE overview](#)

Walkthrough: LinqToXmlDataBinding example

2/8/2019 • 2 minutes to read • [Edit Online](#)

This walkthrough describes the LinqToXmlDataBinding example, and explains some of the more interesting contents of its two primary source files, *L2DBForm.xaml* and *L2DBForm.xaml.cs*.

Prerequisites

Before you read this walkthrough, we highly recommended that you build and run the LinqToXmlDataBinding program as described in [How to: Build and run the LinqToXmlDataBinding example](#).

Remarks

The LinqToXmlDataBinding program is a Windows Presentation Foundation (WPF) application that is composed of C# and XAML source files. It contains an embedded XML document that defines a list of books, and enables the user to view, add, delete, and edit these entries. It is composed of the following two primary source files:

- *L2DBForm.xaml* contains the XAML declaration code for the user interface (UI) of the main window. It also contains a window resource section that defines a data provider and embedded XML document for the book listings.
- *L2DBForm.xaml.cs* contains the initialization and event-handling methods associated with the UI.

The main window is divided into the following four vertical UI sections:

- **XML** displays the raw XML source of the embedded book listing.
- **Book List** displays the book entries as standard text and enables the user to select and delete individual entries.
- **Edit Selected Book** enables the user to edit the values associated with the currently selected book entry.
- **Add New Book** enables the creation of a new book entry based on values entered by the user.

In this section

TOPIC	DESCRIPTION
L2DBForm.xaml source code	Contains the contents and description of the XAML code in file <i>L2DBForm.xaml</i> .
L2DBForm.xaml.cs source code	Contains the contents and description of the C# source code in the file <i>L2DBForm.xaml.cs</i> .

See also

- [WPF data binding using LINQ to XML example](#)
- [How to: Build and run the LinqToXmlDataBinding example](#)

L2DBForm.xaml source code

2/8/2019 • 4 minutes to read • [Edit Online](#)

This topic contains and describes the XAML source file for the [WPF data binding using LINQ to XML example](#), *L2DBForm.xaml*.

Overall UI structure

As is typical for a WPF project, this file contains one parent element, a [Window](#) XML element associated with the derived class `L2XDBFrom` in the `LinqToXmlDataBinding` namespace.

The client area is contained within a [StackPanel](#) that is given a light blue background. This panel contains four [DockPanel](#) UI sections separated by [Separator](#) bars. The purpose of these sections is described in the **Remarks** in the [previous topic](#).

Each section contains a label that identifies it. In the first two sections, this label is rotated 90 degrees through the use of a [LayoutTransform](#). The rest of the section contains UI elements appropriate to the purpose of that section: text blocks, text boxes, buttons, and so on. Sometimes a child [StackPanel](#) is used to align these child controls.

Window resource section

The opening `<Window.Resources>` tag on line 9 indicates the start of the window resource section. It ends with the closing tag on line 35.

The `<ObjectDataProvider>` tag, which spans lines 11 through 25, declares a [ObjectDataProvider](#), named `LoadedBooks`, that uses an [XElement](#) as the source. The [XElement](#) is initialized by parsing an embedded XML document (a `<CDATA>` element). Notice that white space is preserved when declaring the embedded XML document, and also when it's parsed. White space is preserved because the [TextBlock](#) control, which is used to display the raw XML, has no special XML formatting capabilities.

Lastly, a [DataTemplate](#) named `BookTemplate` is defined on lines 28 through 34. This template is used to display the entries in the **Book List** UI section. It uses data binding and LINQ to XML dynamic properties to retrieve the book ID and book name through the following assignments:

```
Text="{Binding Path=Attribute[id].Value}"Text="{Binding Path=Value}"
```

Data binding code

In addition to the [DataTemplate](#) element, data binding is used in a number of other places in this file.

In the opening `<StackPanel>` tag on line 38, the [DataContext](#) property of this panel is set to the `LoadedBooks` data provider.

```
DataContext="{Binding Source={StaticResource LoadedBooks}}
```

Setting the data context makes it possible (on line 46) for the [TextBlock](#) named `tbRawXml` to display the raw XML by binding to this data provider's `Xml` property:

```
Text="{Binding Path=Xml}"
```

The [ListBox](#) in the **Book List** UI section, on lines 58 through 62, sets the template for its display items to the

`BookTemplate` defined in the window resource section:

```
ItemTemplate="{StaticResource BookTemplate}"
```

Then, on lines 59 through 62, the actual values of the books are bound to this list box:

```
<ListBox.ItemsSource>
    <Binding Path="Elements[{http://www.mybooks.com}book]"/>
</ListBox.ItemsSource>
```

The third UI section, **Edit Selected Book**, first binds the [DataContext](#) of the parent [StackPanel](#) to the currently selected item in from the **Book List** UI section (line 82):

```
DataContext="{Binding ElementName=lbBooks, Path=SelectedItem}"
```

It then uses two-way data binding, so that the current values of the book elements are displayed to, and updated from, the two text boxes in this panel. Data binding to dynamic properties is similar to the data binding used in the

`BookTemplate` data template:

```
Text="{Binding Path=Attribute[id].Value}"...Text="{Binding Path=Value}"
```

The last UI section, **Add New Book**, doesn't use data binding in its XAML code. Instead, data binding is in its event handling code in the file *L2DBForm.xaml.cs*.

Example

Description

NOTE

We recommend that you copy the following code below into a code editor, such as the C# source code editor in Visual Studio, so that line numbers will be easier to track.

Code

```
<Window x:Class="LinqToXmlDataBinding.L2XDBForm"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    xmlns:xinq="clr-namespace:System.Xml.Linq;assembly=System.Xml.Linq"
    xmlns:local="clr-namespace:LinqToXmlDataBinding"
    Title="WPF Data Binding using LINQ-to-XML" Height="665" Width="500" ResizeMode="NoResize">

    <Window.Resources>
        <!-- Books provider and inline data -->
        <ObjectDataProvider x:Key="LoadedBooks" ObjectType="{x:Type xinq:XElement}" MethodName="Parse">
            <ObjectDataProvider.MethodParameters>
                <system:String xml:space="preserve">
<![CDATA[
<books xmlns="http://www.mybooks.com">
    <book id="0">book zero</book>
    <book id="1">book one</book>
    <book id="2">book two</book>
    <book id="3">book three</book>
</books>
]]>
            </system:String>
        </ObjectDataProvider.MethodParameters>
        </ObjectDataProvider>
    </Window.Resources>
```



```

]]>
        </system:String>
        <xling:LoadOptions>PreserveWhitespace</xling:LoadOptions>
    </ObjectDataProvider.MethodParameters>
</ObjectDataProvider>

<!-- Template for use in Books List listbox. -->
<DataTemplate x:Key="BookTemplate">
    <StackPanel Orientation="Horizontal">
        <TextBlock Margin="3" Text="{Binding Path=Attribute[id].Value}"/>
        <TextBlock Margin="3" Text="-"/>
        <TextBlock Margin="3" Text="{Binding Path=Value}"/>
    </StackPanel>
</DataTemplate>
</Window.Resources>

<!-- Main visual content container -->
<StackPanel Background="lightblue" DataContext="{Binding Source={StaticResource LoadedBooks}}">
    <!-- Raw XML display section -->
    <DockPanel Margin="5">
        <Label Background="Gray" FontSize="12" BorderBrush="Black" BorderThickness="1"
FontWeight="Bold">XML
        <Label.LayoutTransform>
            <RotateTransform Angle="90"/>
        </Label.LayoutTransform>
        </Label>
        <TextBlock Name="tbRawXml" Height="200" Background="LightGray" Text="{Binding Path=Xml}"
TextTrimming="CharacterEllipsis" />
    </DockPanel>

    <Separator Height="4" Margin="5" />

    <!-- List box to display all books section -->
    <DockPanel Margin="5">
        <Label Background="Gray" FontSize="12" BorderBrush="Black" BorderThickness="1"
FontWeight="Bold">Book List
        <Label.LayoutTransform>
            <RotateTransform Angle="90"/>
        </Label.LayoutTransform>
        </Label>
        <ListBox Name="lbBooks" Height="200" Width="415" ItemTemplate="{StaticResource BookTemplate}">
            <ListBox.ItemsSource>
                <Binding Path="Elements[{http://www.mybooks.com}book]"/>
            </ListBox.ItemsSource>
        </ListBox>
        <Button Margin="5" DockPanel.Dock="Right" Height="30" Width="130" Content="Remove Selected Book"
Click="OnRemoveBook">
            <Button.LayoutTransform>
                <RotateTransform Angle="90"/>
            </Button.LayoutTransform>
        </Button>
    </DockPanel>

    <Separator Height="4" Margin="5" />

    <!-- Edit current selection section -->
    <DockPanel Margin="5">
        <TextBlock Margin="5" Height="30" Width="65" DockPanel.Dock="Right" Background="LightGray"
TextWrapping="Wrap" TextAlignment="Center">
            Changes are live!
        <TextBlock.LayoutTransform>
            <RotateTransform Angle="90"/>
        </TextBlock.LayoutTransform>
        </TextBlock>
    </DockPanel>
    <StackPanel>
        <Label Width="450" Background="Gray" FontSize="12" BorderBrush="Black" BorderThickness="1"
HorizontalAlignment="Left" FontWeight="Bold">Edit Selected Book</Label>
        <StackPanel Margin="1" DataContext="{Binding ElementName=lbBooks, Path=SelectedItem}">
            <StackPanel Orientation="Horizontal">

```

```

        <Label Width="40">ID:</Label>
        <TextBox Name="editAttributeTextBox" Width="410" Text="{Binding
Path=Attribute[id].Value}">
            <TextBox.ToolTip>
                <TextBlock FontWeight="Bold" TextAlignment="Center">
                    <Label>Edit the selected book ID and see it changed.</Label>
                </TextBlock>
            </TextBox.ToolTip>
        </TextBox>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="40">Value:</Label>
        <TextBox Name="editValueTextBox" Width="410" Text="{Binding Path=Value}" Height="25">
            <TextBox.ToolTip>
                <TextBlock FontWeight="Bold" TextAlignment="Center">
                    <Label>Edit the selected book Value and see it changed.</Label>
                </TextBlock>
            </TextBox.ToolTip>
        </TextBox>
    </StackPanel>
</StackPanel>
</StackPanel>
</DockPanel>

<Separator Height="4" Margin="5" />

<!-- Add new book section -->
<DockPanel Margin="5">
    <Button Margin="5" Height="30" DockPanel.Dock="Right" Click ="OnAddBook">Add Book
        <Button.LayoutTransform>
            <RotateTransform Angle="90"/>
        </Button.LayoutTransform>
    </Button>
    <StackPanel>
        <Label Width="450" Background="Gray" FontSize="12" BorderBrush="Black" BorderThickness="1"
HorizontalAlignment="Left" FontWeight="Bold">Add New Book</Label>
        <StackPanel Margin="1">
            <StackPanel Orientation="Horizontal">
                <Label Width="40">ID:</Label>
                <TextBox Name="tbAddID" Width="410">
                    <TextBox.ToolTip>
                        <TextBlock FontWeight="Bold" TextAlignment="Center">
                            <Label>Enter a book ID and Value pair, then click Add Book.</Label>
                        </TextBlock>
                    </TextBox.ToolTip>
                </TextBox>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <Label Width="40">Value:</Label>
                <TextBox Name="tbAddValue" Width="410" Height="25">
                    <TextBox.ToolTip>
                        <TextBlock FontWeight="UltraBold" TextAlignment="Center">
                            <Label>Enter a book ID and Value pair, then click Add Book.</Label>
                        </TextBlock>
                    </TextBox.ToolTip>
                </TextBox>
            </StackPanel>
        </StackPanel>
    </StackPanel>
</DockPanel>
</StackPanel>
</Window>

```

Comments

For the C# source code for the event handlers associated with the WPF UI elements, see [L2DBForm.xaml.cs source code](#).

See also

- [Walkthrough: LinqToXmlDataBinding example](#)
- [L2DBForm.xaml.cs source code](#)

L2DBForm.xaml.cs source code

2/8/2019 • 2 minutes to read • [Edit Online](#)

This topic contains the contents and description of the C# source code in the file *L2DBForm.xaml.cs*. The L2XDBForm partial class contained in this file can be divided into three logical sections: data members and the `OnRemove` and `OnAddBook` button click event handlers.

Data members

Two private data members are used to associate this class to the window resources used in *L2DBForm.xaml*.

- The namespace variable `myBooks` is initialized to `"http://www.mybooks.com"`.
- The member `bookList` is initialized in the constructor to the CDATA string in *L2DBForm.xaml* with the following line:

```
bookList = (XElement)((ObjectDataProvider)Resources["LoadedBooks"]).Data;
```

OnAddBook event handler

This method contains the following three statements:

- The first conditional statement is used for input validation.
- The second statement creates a new `XElement` from the string values the user entered in the **Add New Book** user interface (UI) section.
- The last statement adds this new book element to the data provider in *L2DBForm.xaml*. Consequently, dynamic data binding will automatically update the UI with this new item; no extra user-supplied code is required.

OnRemove event handler

The `OnRemove` handler is more complicated than the `OnAddBook` handler for two reasons. First, because the raw XML contains preserved white space, matching newlines must also be removed with the book entry. Second, as a convenience, the selection, which was on the deleted item, is reset to the previous one in the list.

However, the core work of removing the selected book item is accomplished by only two statements:

- First, the book element associated with the currently selected item in the list box is retrieved:

```
XElement selBook = (XElement)lbBooks.SelectedItem;
```

- Then, this element is deleted from the data provider:

```
selBook.Remove();
```

Again, dynamic data binding assures that the program's UI is automatically updated.

Example

Code

```
using System;
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Input;
using System.Xml;
using System.Xml.Linq;

namespace LinqToXmlDataBinding {
    /// <summary>
    /// Interaction logic for L2XDBForm.xaml
    /// </summary>

    public partial class L2XDBForm : System.Windows.Window
    {
        XNamespace mybooks = "http://www.mybooks.com";
        XElement bookList;

        public L2XDBForm()
        {
            InitializeComponent();
            bookList = (XElement)((ObjectDataProvider)Resources["LoadedBooks"]).Data;
        }

        void OnRemoveBook(object sender, EventArgs e)
        {
            int index = lbBooks.SelectedIndex;
            if (index < 0) return;

            XElement selBook = (XElement)lbBooks.SelectedItem;
            //Get next node before removing element.
            XNode nextNode = selBook.NextNode;
            selBook.Remove();

            //Remove any matching newline node.
            if (nextNode != null && nextNode.ToString().Trim().Equals(""))
            { nextNode.Remove(); }

            //Set selected item.
            if (lbBooks.Items.Count > 0)
            { lbBooks.SelectedItem = lbBooks.Items[index > 0 ? index - 1 : 0]; }
        }

        void OnAddBook(object sender, EventArgs e)
        {
            if (String.IsNullOrEmpty(tbAddID.Text) ||
                String.IsNullOrEmpty(tbAddValue.Text))
            {
                MessageBox.Show("Please supply both a Book ID and a Value!", "Entry Error!");
                return;
            }
            XElement newBook = new XElement(
                mybooks + "book",
                new XAttribute("id", tbAddID.Text),
                tbAddValue.Text);
            bookList.Add(" ", newBook, "\r\n");
        }
    }
}
```

Comments

For the associated XAML source for these handlers, see [L2DBForm.xaml source code](#).

See also

- [Walkthrough: LinqToXmlDataBinding example](#)
- [L2DBForm.xaml source code](#)

Work with 3D assets for games and apps

2/8/2019 • 3 minutes to read • [Edit Online](#)

This document describes the Visual Studio tools that you can use to create or modify 3D models, textures, and shaders for DirectX-based games and apps.

DirectX app development in Visual Studio

A DirectX app typically combines programming logic, the DirectX API, and High Level Shading Language (HLSL) programs, together with audio and 3D visual assets to present a rich, interactive multimedia experience. Visual Studio includes tools that you can use to work with images and textures, 3D models, and shaders without leaving the IDE to use another tool. The Visual Studio tools are especially suited for creating *placeholder* assets, which you can use to test code or build prototypes before you commission production-ready assets, and for inspecting and modifying production-ready assets when you are debugging your app.

Here's more information about the kinds of assets you can work with in Visual Studio.

Images and textures

Images and textures provide color and visual detail in games and apps. In 3D graphics, textures come in a variety of formats, types, and geometries to support different uses. For example, normal maps provide per-pixel surface normals for more-detailed lighting of 3D models, and cube maps provide texture in all directions for uses such as sky-boxing, reflections, and spherical texture mapping. Textures can provide mipmaps to support efficient rendering at different levels of detail, and can support different color channels and color orderings. Textures can be stored in a variety of compressed formats that occupy less dedicated graphics memory and help GPUs access textures more efficiently.

You can use the Visual Studio Image Editor to work with images and textures in many common types and formats.

3D models

3D models create space and shape in games and apps. Minimally, models encode the position of points in 3D space—which are known as *vertices*—together with indexing data to define lines or triangles that represent the shape of the model. Additional data can be associated with these vertices—for example, color information, normal vectors, or application-specific attributes. Each model can also define object-wide attributes—for example, which shader is used to compute the appearance of the object's surface, or which texture is applied to it.

You can use the Visual Studio Model Editor to work with 3D models in several common formats.

Shaders

Shaders are small, domain-specific programs that run on the graphics processing unit (GPU). Shaders determine how 3D models are transformed into on-screen shapes and how each pixel in those shapes is colored. By creating a shader and applying it to an object in your game or app, you can give the object a unique appearance.

You can use the Visual Studio Shader Designer, which is a graph-based shader design tool, to create custom visual effects without knowing HLSL programming.

NOTE

For more information about how to start with DirectX programming, see [DirectX](#). For more information about how to debug a DirectX-based app, see [Graphics diagnostics \(debugging DirectX graphics\)](#).

DirectX version compatibility

Visual Studio uses DirectX to render 2D and 3D assets. You can select either the DirectX 11 renderer, or the Windows Advanced Rasterization Platform (WARP) software renderer. The DirectX 11 renderer provides high-performance, hardware-accelerated rendering on DirectX 11 and DirectX 10 GPUs. The WARP renderer helps make sure that your assets work with a broad range of computers—this includes computers that don't have modern graphics hardware and computers that have integrated graphics hardware. For more information about WARP, see [Windows Advanced Rasterization Platform \(WARP\) guide](#).

Related topics

TITLE	DESCRIPTION
Working with textures and images	Describes how to use Visual Studio to work with images and textures.
Working with 3D models	Describes how to use Visual Studio to work with 3D models.
Working with shaders	Describes how to use the Visual Studio Shader Designer to create and modify custom shader effects.
Using 3D assets in your game or app	Describes how to use assets, which you created by using the Image Editor, Model Editor, or Shader Designer, in your game or app.

Work with textures and images

2/8/2019 • 2 minutes to read • [Edit Online](#)

You can use the Image Editor in Visual Studio to create and modify textures and images. The Image Editor supports rich texture and image formats like those that are used in DirectX app development.

NOTE

The Image Editor doesn't support low-color images like icons or cursors. To create or modify those kinds of images, use the [Image Editor for icons \(C++\)](#).

Textures and images

Textures and images are, at a basic level, just tables of data that are used to provide visual detail in graphics apps. The kind of detail that a texture or image provides depends on how it's used, but color samples, alpha (transparency) values, surface normals, and height values are common examples. The primary difference between a texture and an image is that a texture is meant to be used together with a representation of shape—typically a 3D model—to express a complete object or scene, but an image is typically a stand-alone representation of the object or scene.

Any texture can be encoded and compressed in a number of ways that are orthogonal to the type of data that a texture holds, or to the dimensionality or "shape" of the texture. However, different encoding and compression methods yield better results for different kinds of data.

You can use the Image Editor to create and modify textures and images in ways that resemble other image editors. The Image Editor also provides mipmapping and other features for use with 3D graphics, and supports many of the highly-compressed, hardware-accelerated texture formats that DirectX supports.

Common kinds of textures include:

Texture maps

Texture maps contain color values that are organized as a one-, two-, or three-dimensional matrix. They are used to provide color detail on the affected object. Colors are commonly encoded by using RGB (red, green, blue) color channels, and may include a fourth channel, alpha, that represents transparency. Less commonly, colors could be encoded in another color scheme, or the fourth channel could contain data other than alpha—for example, height.

Normal maps

Normal maps contain surface normals. They are used to provide lighting detail on the affected object. Normals are commonly encoded by using the red, green, and blue color components to store the x, y, and z dimensions of the vector. However, other encodings exist—for example, encodings that are based on polar coordinates.

Height maps

Height maps contain height-field data. They are used to provide a form of geometric detail on the affected object—by using shader code to compute the desired effect—or to provide data points for uses like terrain generation. Height values are commonly encoded by using one channel in a texture.

Cube maps

Cube maps can contain different types of data—for example, colors or normals—but are organized as six textures on the faces of a cube. Because of this, cube maps are not sampled by supplying texture coordinates, but by supplying a vector whose origin is the center of the cube; the sample is taken at the point where the vector

intersects the cube. Cube maps are used to provide an approximation of the environment that can be used to calculate reflections—this is known as *environment mapping*—or to provide texture to spherical objects with less distortion than basic, two-dimensional textures can provide.

Related topics

TITLE	DESCRIPTION
Image Editor	Describes how to use the Image Editor to work with textures and images.
Image Editor examples	Provides links to topics that demonstrate how to use the Image Editor to perform common image processing tasks.

Image editor

2/8/2019 • 18 minutes to read • [Edit Online](#)

This article describes how to work with the Visual Studio **Image Editor** to view and modify texture and image resources.

You can use the **Image Editor** to work with the kinds of rich texture and image formats that are used in DirectX app development. This includes support for popular image file formats and color encodings, features such as alpha-channels and MIP-mapping, and many of the highly compressed, hardware-accelerated texture formats that DirectX supports.

Supported formats

The **Image Editor** supports the following image formats:

FORMAT NAME	FILE NAME EXTENSION
Portable Network Graphics	<i>.png</i>
JPEG	<i>.jpg, .jpeg, .jpe, .jif</i>
Direct Draw Surface	<i>.dds</i>
Graphics Interchange Format	<i>.gif</i>
Bitmap	<i>.bmp, .dib</i>
Tagged Image File Format	<i>.tif, .tiff</i>
TGA (Targa)	<i>.tga</i>

Get started

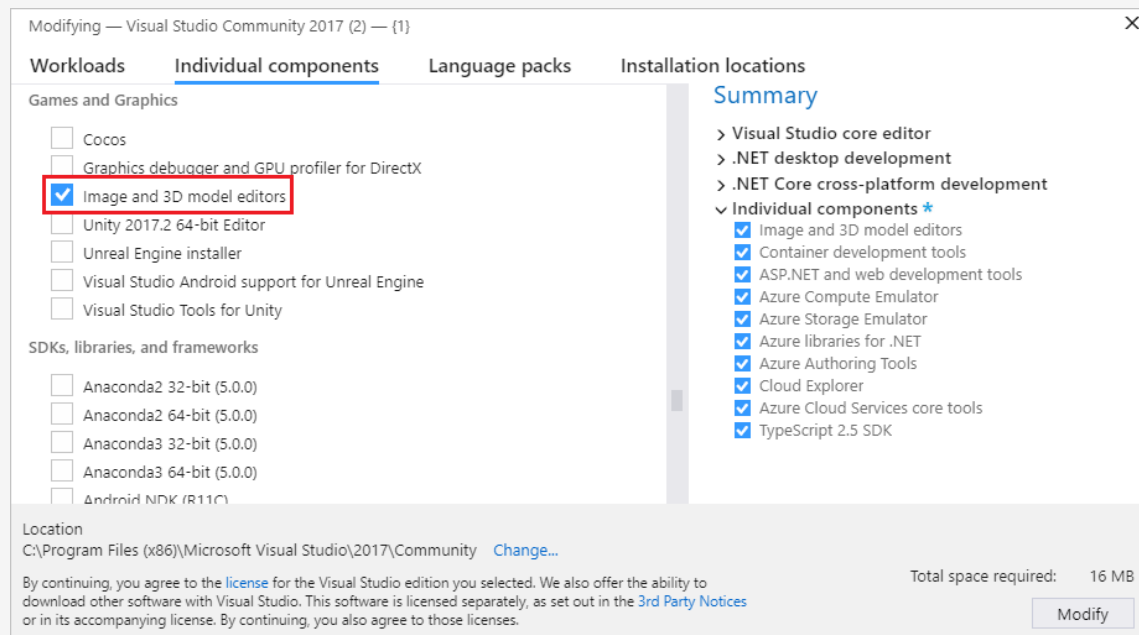
This section describes how to add an image to your Visual Studio project and configure it for your requirements.

Add an image to your project

1. In **Solution Explorer**, open the shortcut menu for the project that you want to add the image to, and then choose **Add > New Item**.
2. In the **Add New Item** dialog box, under **Installed**, select **Graphics**, and then select an appropriate file format for the image.

NOTE

If you don't see the **Graphics** category in the **Add New Item** dialog, you may need to install the **Image and 3D model editors** component. Close the dialog and then select **Tools > Get Tools and Features** from the menu bar, to open the **Visual Studio Installer**. Select the **Individual components** tab, and then select the **Image and 3D model editors** component under the **Games and Graphics** category. Select **Modify**.



For information about how to choose a file format based on your requirements, see [Choose the image format](#).

3. Specify the **Name** of the image file and the **Location** where you want it to be created.
4. Choose the **Add** button.

Choose the image format

Depending on how you plan to use the image, certain file formats might be more appropriate than others. For example, some formats might not support a feature that you need, for example, transparency or a specific color format. Some formats might not provide suitable compression for the kind of image content you have planned.

The following information can help you choose an image format that meets your needs:

Bitmap Image (.bmp)

The bitmap image format. An uncompressed image format that supports 24-bit color. The bitmap format doesn't support transparency.

GIF Image (.gif)

The Graphics Interchange Format (GIF) image format. An LZW-compressed, lossless image format that supports up to 256 colors. Unsuitable for photographs and images that have a significant amount of color detail, but provides good compression ratios for low-color images that have a high degree of color coherence.

JPG Image (.jpg)

The Joint Photographic Experts Group (JPEG) image format. A highly compressed, lossy image format that supports 24-bit color and is suitable for general-purpose compression of images that have a high degree of color coherence.

PNG Image (.png)

The Portable Network Graphics (PNG) image format. A moderately compressed, lossless image format that supports 24-bit color and alpha transparency. It is suitable for both natural and artificial images, but does not provide compression ratios as good as lossy formats such as JPG or GIF.

TIFF Image (.tif)

The Tagged Image File Format (TIFF or TIF) image format. A flexible image format that supports several compression schemes.

DDS Texture (.dds)

The DirectDraw Surface (DDS) texture format. A highly compressed, lossy texture format that supports 24-bit color and alpha transparency. Its compression ratios can be as high as 8:1. It's based on S3 Texture compression, which can be decompressed on graphics hardware.

TGA Image (.tga)

The Truevision Graphics Adapter (TGA) image format (also known as Targa). An RLE-compressed, lossless image format that supports both color-mapped (color palette) or direct-color images of up to 24-bit color and alpha transparency. Unsuitable for photographs and images that have a significant amount of color detail, but provides good compression ratios for images that have long spans of identical colors.

Configure the image

Before you begin to work with the image that you created, you can change its default configuration. For example, you can change its dimensions or the color format that it uses. For information about how to configure these and other properties of the image, see [Image properties](#).

NOTE

Before you save your work, make sure to set the **Color Format** property if you want to use a specific color format. If the file format supports compression, you can adjust the compression settings when you save the file for the first time or when you choose **Save As**.

Work with the Image Editor

This section describes how to use the **Image Editor** to modify textures and images.

Commands that affect the state of the **Image Editor** are located on the **Image Editor Mode** toolbar together with advanced commands. The toolbar is located along the topmost edge of the **Image Editor** design surface. Drawing tools are located on the **Image Editor** toolbar along the leftmost edge of the **Image Editor** design surface.

Image Editor Mode toolbar



The following table describes the items on the **Image Editor Mode** toolbar, which are listed in the order in which they appear from left to right:

TOOLBAR ITEM	DESCRIPTION
Select	Enables selection of a rectangular region of an image. After you select a region, you can cut, copy, move, scale, rotate, flip, or delete it. When there is an active selection, drawing tools only affect the selected region.

TOOLBAR ITEM	DESCRIPTION
Irregular Selection	Enables selection of an irregular region of an image. After you select a region, you can cut, copy, move, scale, rotate, flip, or delete it. When there is an active selection, drawing tools only affect the selected region.
Wand Selection	Enables selection of a similarly colored region of an image. The <i>tolerance</i> —that is, the maximum difference between adjacent colors within which they are considered similar—can be configured to include a smaller or wider range of similar colors. After you select a region, you can cut, copy, move, scale, rotate, flip, or delete it. When there is an active selection, drawing tools only affect the selected region.
Pan	<p>Enables movement of the image relative to the window frame. In Pan mode, select a point on the image and then move it around.</p> <p>You can temporarily activate Pan mode by pressing and holding the Ctrl key.</p>
Zoom	<p>Enables the display of more or less image detail relative to the window frame. In Zoom mode, select a point on the image and then move it right or down to zoom in, or left or up to zoom out.</p> <p>You can zoom in or out by pressing and holding Ctrl while you either use the mouse wheel or press the plus sign (+) or minus sign (-).</p>
Zoom to Actual Size	Displays the image by using a 1:1 relationship between the pixels of the image and the pixels of the screen.
Zoom To Fit	Displays the full image in the window frame.
Zoom To Width	Displays the full width of the image in the window frame.
Grid	Enables or disables the grid that shows pixel boundaries. The grid might not appear until you zoom into the image.
View Next MIP Level	Activates the next larger MIP level in a MIP map chain. The active MIP level is displayed on the design surface. This item is only available for textures that have MIP levels.
View Previous MIP Level	Activates the next smaller MIP level in a MIP map chain. The active MIP level is displayed on the design surface. This item is only available for textures that have MIP levels.
Red Channel Green Channel Blue Channel Alpha Channel	Enables or disables the specific color channel. Note: By systematically enabling or disabling color channels, you can isolate problems that are related to one or more of them. For example, you could identify incorrect alpha transparency.

TOOLBAR ITEM	DESCRIPTION
<p>Background</p>	<p>Enables or disables display of the background through transparent parts of the image. You can configure how the background is displayed by choosing from these options:</p> <p>Checkerboard Uses a green color together with the specified background color to display the background as a checkerboard pattern. You can use this option to help make transparent parts of the image more apparent.</p> <p>White Background Uses the color white to display the background.</p> <p>Black Background Uses the color black to display the background.</p> <p>Animate Background Pans the checkerboard pattern slowly. You can use this option to help make transparent parts of the image more apparent.</p>
<p>Properties</p>	<p>Alternately opens or closes the Properties window.</p>

TOOLBAR ITEM	DESCRIPTION
Advanced	<p>Contains additional commands and options.</p> <p>Filters</p> <p>Provides several common image filters: Black and White, Blur, Brighten, Darken, Edge Detection, Emboss, Invert Colors, Ripple, Sepia Tone, and Sharpen.</p> <p>Graphics Engines</p> <p>Render with D3D11 Uses Direct3D 11 to render the Image Editor design surface.</p> <p>Render with D3D11WARP Uses Direct3D 11 Windows Advanced Rasterization Platform (WARP) to render the Image Editor design surface.</p> <p>Tools</p> <p>Flip Horizontal Transposes the image around its horizontal, or x, axis.</p> <p>Flip Vertical Transposes the image around its vertical, or y, axis.</p> <p>Generate Mips Generates MIP levels for an image. If MIP levels already exist, they are recreated from the largest MIP level. Any changes that were made to smaller MIP levels are lost. To save the MIP levels that you have generated, you must use the <i>.dds</i> format to save the image.</p> <p>View</p> <p>Frame Rate When enabled, displays the frame rate in the upper-right corner of the design surface. The frame rate is the number of frames that are drawn per second. Tip: You can choose the Advanced button to run the last command again.</p>

Image Editor toolbar



The following table describes the items on the **Image Editor** toolbar, which are listed in the order in which they appear from top to bottom:

TOOLBAR ITEM	DESCRIPTION
Pencil	Uses the active color selection to draw an aliased stroke. You can set the color and thickness of the stroke in the Properties window.
Brush	Uses the active color selection to draw an anti-aliased stroke. You can set the color and thickness of the stroke in the Properties window.
Airbrush	Uses the active color selection to draw an anti-aliased stroke that blends together with the image and becomes more saturated as a function of time. You can set the color and thickness of the stroke in the Properties window.
Eyedropper	Sets the active color selection to the color of the selected pixel.
Fill	<p>Uses the active color selection to fill a region of the image. The affected region is defined as the pixel where the fill is applied, together with every pixel that is connected to it by pixels of the same color and that is the same color itself. If the fill is applied within an active selection, then the affected region is constrained by the selection.</p> <p>By default, the active color selection is blended together with the affected region of the image according to its alpha component. To use the active color selection to overwrite the affected region, press and hold the Shift key when you use the fill tool.</p>
Eraser	Sets pixels to the fully transparent color if the image supports an alpha channel. Otherwise, sets the pixels to the active background color.
Line, Rectangle, Rounded Rectangle, Ellipse	<p>Draws a shape on the image. You can set the color and thickness of the outline in the Properties window.</p> <p>To draw a primitive that has equal width and height, press and hold Shift as you draw.</p>
Text	Uses the foreground color selection to draw text. The background color is determined by the background color selection. For a transparent background, the alpha value of the background color selection must be 0. While the text region is active, you can set whether the text is drawn with an anti-aliased stroke, and you can set the text Value , Font , Size , and style— Bold , Italics , or Underlined —in the Properties window. The content and appearance of the text is finalized when the text region is no longer active.
Rotate	Rotates the image 90 degrees clockwise.
Trim	Trims the image to the active selection.

Work with MIP levels

Some image formats, for example, DirectDraw Surface (.dds), support MIP levels for texture-space Level-of-Detail (LOD). For information about how to generate and work with MIP levels, see [How to: Create and modify](#)

Work with transparency

Some image formats, for example, DirectDraw Surface (.dds), support transparency. There are several ways you can use transparency, depending on the tool that you're using. To specify the level of transparency for a color selection, in the **Properties** window, set the **A** (alpha) component of the color selection.

The following table describes how different kinds of tools control how transparency is applied:

TOOL	DESCRIPTION
Pencil, Brush, Airbrush, Line, Rectangle, Rounded Rectangle, Ellipse, Text	<p>To blend the active color selection together with the image, in the Properties window, expand the Channels property group and set the Draw checkbox on the Alpha channel, and then draw normally.</p> <p>To draw by using the active color selection and leave the alpha value of the image in place, clear the Draw checkbox of the Alpha channel, and then draw normally.</p>
Fill	<p>To blend the active color selection together with the image, just choose the area to fill.</p> <p>To use the active color selection—including the value of the alpha channel—to overwrite the image, press and hold Shift and then choose the area to fill.</p>

Image properties

You can use the **Properties** window to specify various properties of the image. For example, you can set the width and height properties to resize the image.

The following table describes image properties:

PROPERTY	DESCRIPTION
Width	The width of the image.
Height	The height of the image.
Bits Per Pixel	The number of bits that represent each pixel. The value of this property depends on the Color Format of the image.
Transparent Selection	True to blend the selection layer together with the main image, based on the alpha value of the selection layer; otherwise, False . This item is only available for images that support alpha.
Format	The color format of the image. You can specify a variety of color formats, depending on the image format. The color format defines the number and kind of color channels that are included in the image, and also the size and encoding of various channels.
Mip Level	The active MIP level. This item is only available for textures that have MIP levels.

PROPERTY	DESCRIPTION
Mip Level Count	The total number of MIP levels in the image. This item is only available for textures that have MIP levels.
Frame Count	The total number of frames in the image. This item is only available for images that support texture arrays.
Frame	The current frame. Only the first frame can be viewed; all other frames are lost when the image is saved.
Depth Slice Count	The total number of depth slices in the image. This item is only available for images that support volume textures.
Depth Slice	The current depth slice. Only the first slice can be viewed; all other slices are lost when you save the image.

NOTE

Because the **Rotate by** property applies to all tools and selected regions, it always appears at the bottom of the **Properties** window together with other tool properties. **Rotate by** is always displayed because the whole image is implicitly selected when there is no other selection or active tool. For more information about the **Rotate by** property, see [Tool properties](#tool-properties).

Resize images

There are two ways to resize an image. In both cases, the **Image Editor** uses bilinear interpolation to resample the image.

- In the **Properties** window, specify new values for the **Width** and **Height** properties.
- Select the entire image and use the border markers to resize the image.

Selected regions

Selections in the **Image Editor** define regions of the image that are active. Active regions are affected by tools and transformations. When there is an active selection, areas outside the selected region are not affected by most tools and transformations. If there is not an active selection, the entire image is active.

Most tools (**Pencil**, **Brush**, **Airbrush**, **Fill**, **Eraser**, and 2D primitives) and transformations (**Rotate**, **Trim**, **Invert Colors**, **Flip Horizontal**, and **Flip Vertical**) are constrained or defined by the active selection. However, some tools (**Eyedropper** and **Text**) and transformations (**Generate Mips**) aren't affected by any active selection. These tools always behave as if the entire image is the active selection.

While you're selecting a region, you can press and hold **Shift** to make a proportional (square) selection. Otherwise, the selection is not constrained.

Resize selections

After you select a region, you can resize it or its image contents by changing the size of the selection marker. While you're resizing the selected region, you can use the following modifier keys to change the behavior of the selected region as you resize it:

Ctrl - Copies the contents of the selected region before it's resized. This leaves the original image intact while the copy is resized.

Shift - Resizes the selected region in proportion to its original size.

Alt - Changes the size of the selection region. This leaves the image unmodified.

The following table describes the valid modifier key combinations:

CTRL	SHIFT	ALT	DESCRIPTION
			Resizes the content of the selected region.
	Shift		Proportionally resizes the content of the selected region.
		Alt	Resizes the selected region. This defines a new selection region.
	Shift	Alt	Proportionally resizes the selected region. This defines a new selection region.
Ctrl			Copies and then resizes the content of the selected region.
Ctrl	Shift		Copies and then proportionally resizes the content of the selected region.

Tool properties

While a tool is selected, you can use the **Properties** window to specify details about how it affects the image. For example, you can set the thickness of the **Pencil** tool or the color of the **Brush** tool.

You can set both a foreground color and a background color. Both support an alpha channel to provide user-defined opacity. The settings apply to all tools. If you use a mouse, the left mouse button corresponds to the foreground color, and the right mouse button corresponds to the background color.

The following table describes tool properties:

TOOL	PROPERTIES
All tools and selections	Rotate by Defines the amount, in degrees, that the selection or tool effect is rotated in the clockwise direction.
Pencil, Brush, Airbrush, Eraser	Thickness Defines the size of the area that is affected by the tool.

TOOL	PROPERTIES
Text	<p>Anti-alias Draws text that has anti-aliased edges. This gives text a smoother appearance.</p> <p>Value The text to be drawn.</p> <p>Font The font used to draw the text.</p> <p>Size The size of the text.</p> <p>Bold Makes the font bold.</p> <p>Italics Makes the font italic.</p> <p>Underlined Makes the font underlined.</p>
2D Primitive	<p>Anti-alias Draws primitives that have anti-aliased edges. This gives them a smoother appearance.</p> <p>Thickness Defines the thickness of the line that forms the boundary of the primitive.</p> <p>Radius X (Rounded rectangle only) Defines the rounding radius for the top and bottom edges of the primitive.</p> <p>Radius Y (Rounded rectangle only) Defines the rounding radius for the left and right edges of the primitive.</p>
Pencil, Brush, Airbrush, 2D Primitive	<p>Channels Enables or disables specific color channels for viewing and drawing. If View is set for a specific color channel, that channel is visible in the image; otherwise, it is not visible. If Draw is set for a specific color channel, that channel is affected by drawing operations; otherwise, it is not.</p>
Wand Selection, Fill	<p>Tolerance Defines the maximum difference between adjacent colors within which they are considered similar, so that fewer or more similar colors are made a part of the affected or selected region. By default, the value is 32, which means that adjacent pixels within 32 shades (lighter or darker) of the original color are considered to be part of the region.</p>

Keyboard shortcuts

COMMAND	KEYBOARD SHORTCUTS
Switch to Select mode	S

COMMAND	KEYBOARD SHORTCUTS
Switch to Zoom mode	Z
Switch to Pan mode	K
Select all	Ctrl+A
Delete the current selection	Delete
Cancel the current selection	Esc (escape)
Zoom in	Ctrl+Mouse wheel forward Ctrl+PageUp Plus Sign (+)
Zoom out	Ctrl-Mouse wheel backward Ctrl-PageDown Minus Sign (-)
Pan the image up	Mouse wheel backward PageDown
Pan the image down	Mouse wheel forward PageUp
Pan the image left	Shift+ Mouse wheel backward Mouse wheel left Shift+ PageDown
Pan the image right	Shift+ Mouse wheel forward Mouse wheel right Shift+ PageUp
Zoom to actual size	Ctrl+0 (zero)
Fit image to window	Ctrl+G, Ctrl+F
Fit image to window width	Ctrl+G, Ctrl+I
Toggle grid	Ctrl+G, Ctrl+G
Crop image to current selection	Ctrl+G, Ctrl+C
View next (higher detail) MIP level	Ctrl+G, Ctrl+6

COMMAND	KEYBOARD SHORTCUTS
View previous (lower detail) MIP level	Ctrl+G, Ctrl+7
Toggle red color channel	Ctrl+G, Ctrl+1
Toggle green color channel	Ctrl+G, Ctrl+2
Toggle blue color channel	Ctrl+G, Ctrl+3
Toggle alpha (transparency) channel	Ctrl+G, Ctrl+4
Toggle alpha checkerboard pattern	Ctrl+G, Ctrl+B
Switch to irregular selection tool	L
Switch to wand selection tool	M
Switch to pencil tool	P
Switch to brush tool	B
Switch to fill tool	F
Switch to eraser tool	E
Switch to text tool	T
Switch to color-select (eyedropper) tool	I
Move the active selection, and its contents.	Arrow keys.
Resize the active selection, and its contents.	Ctrl+Arrow keys
Move the active selection, but not its contents.	Shift+Arrow keys
Resize the active selection, but not its contents.	Shift+Ctrl+Arrow keys
Commit the current layer	Return
Decrease tool thickness	[
Increase tool thickness]

Related topics

TITLE	DESCRIPTION
Working with 3D assets for games and apps	Provides an overview of the tools that you can use in Visual Studio to work with graphics assets such as textures and images, 3D models, and shader effects.

TITLE	DESCRIPTION
Model editor	Describes how to use the Visual Studio Model Editor to work with 3D models.
Shader designer	Describes how to use the Visual Studio Shader Designer to work with shaders.

Image Editor examples

2/8/2019 • 2 minutes to read • [Edit Online](#)

The articles in this section of the documentation contain examples that demonstrate how you can use the Image Editor.

TITLE	DESCRIPTION
How to: Create a basic texture	Demonstrates how to create a basic texture.
How to: Create and modify MIP levels	Demonstrates how to generate MIP Levels from an image.
Using 3D assets in your game or app	Describes how you can use Visual Studio to process 3D assets when you build your project or solution so that they are ready for use in your app. Strategies for loading different kinds of assets into your app are also discussed.
How to: Export a texture that contains mipmaps	Describes how to use the Image Content Pipeline to export a texture that contains precomputed mipmaps.
How to: Export a texture that has premultiplied alpha	Describes how to use the Image Content Pipeline to export a texture that contains premultiplied alpha values.
How to: Export a texture for use with Direct2D or Javascript apps	Describes how to use the Image Content Pipeline to export a texture that can be used in a Direct2D or Javascript App.

How to: Create a basic texture

2/8/2019 • 3 minutes to read • [Edit Online](#)

This article demonstrates how to use the Image Editor to create a basic texture, including the following activities:

- Setting the size of the texture
- Setting the foreground and background colors
- Using the alpha channel (transparency)
- Using the **Fill** and **Ellipse** tools
- Setting tool properties

Create a basic texture

You can use the Image Editor to create and modify images and textures for your game or app.

The following steps show how to create a texture that represents a "bullseye" target. When you are finished, the texture should look like the following picture. To better demonstrate the transparency in the texture, the Image Editor has been configured to use a green, checkered pattern to display it.



Before you begin, make sure that the **Properties** window is displayed. You use the **Properties** window to set the image size, change tool properties, and specify colors while you work.

Create a "bullseye" target texture

1. Create a texture with which to work. For information about how to add a texture to your project, see [Image Editor](#).
2. Set the image size to 512x512 pixels. In the **Properties** window, set the values of the **Width** and **Height** properties to .
3. On the Image Editor toolbar, choose the **Fill** tool. The **Properties** window now displays the properties of the **Fill** tool together with the image properties.
4. Set the foreground color to fully transparent black. In the **Properties** window, in the **Colors** property group, select **Foreground**. Set the values of the **R**, **G**, **B**, and **A** properties next to the color picker to .
5. On the Image Editor toolbar, choose the **Fill** tool, and then press and hold the **Shift** key and choose any point in the image. Using the **Shift** key causes the alpha value of the fill color to replace the color in the

image; otherwise, the alpha value is used to blend the fill color together with the color in the image.

IMPORTANT

This step, together with the color selection in the previous step, ensures that the base image is prepared for the "bullseye" target texture that you will draw. When the image is filled with transparent black—and because the border of the target is black—there will be no aliasing artifacts around the target.

6. On the Image Editor toolbar, choose the **Ellipse** tool.
7. Set the foreground color to fully opaque black. Set the values of the **R**, **G**, and **B** properties to and the value of the **A** property to .
8. Set the background color to fully opaque white. In the **Properties** window, in the **Colors** property group, select **Background**. Set the values of the **R**, **G**, **B**, and **A** properties to .
9. Set the width of the outline of the ellipse. In the **Properties** window, in the **Appearance** property group, set the value of the **Width** property to .
10. Make sure that anti-aliasing is enabled. In the **Properties** window, in the **Appearance** property group, make sure that the **Anti-alias** property is set.
11. Using the **Ellipse** tool, draw a circle from pixel coordinate to pixel coordinate . To draw the circle more easily, you can press and hold the **Shift** key while you draw.

NOTE

The pixel coordinates of the current pointer location are displayed on the Visual Studio status bar.

12. Change the background color. Set **R** to , **G** to , **B** to , and **A** to .
13. Draw another circle from pixel coordinate to pixel coordinate .
14. Change the background color back to fully opaque white. Set **R**, **G**, **B**, and **A** to .
15. Draw another circle from pixel coordinate to pixel coordinate .
16. Change the background color. Set **R** to , **G** and **B** to , and **A** to .
17. Draw another circle from pixel coordinate to pixel coordinate .

The "bullseye" target texture is complete. Here's the final image, shown with transparency.



As a next step, you can generate MIP levels for this texture. For information, see [How to: Create and modify MIP](#)

[levels](#).

See also

- [Image Editor](#)

How to: Create and modify MIP levels

2/8/2019 • 2 minutes to read • [Edit Online](#)

This document demonstrates how to use the **Image Editor** to generate and modify *MIP levels* for texture-space Level-of-Detail (LoD).

Generating MIP levels

Mipmapping is a technique that's used to increase rendering speed and reduce aliasing artifacts on textured objects by pre-calculating and storing several copies of a texture in different sizes. Each copy, which is known as a MIP level, is half the width and height of the previous copy. When a texture is rendered on the surface of an object, the MIP level that corresponds most closely to the screen-space area of the textured surface is automatically chosen. This means that the graphics hardware doesn't have to filter oversized textures to maintain consistent visual quality. Although the memory cost of storing the MIP levels is about 33 percent more than that of the original texture alone, the performance and image-quality gains justify it.

To generate MIP levels

1. Begin with a basic texture, as described in [How to: Create a basic texture](#). For best results, specify a texture that has a width and height that are a power of two in size, for example, 256, 512, 1024, and so on.
2. Generate the MIP levels. On the **Image Editor Mode** toolbar, choose **Advanced** > **Tools** > **Generate Mips**.

Notice that the **Go to Next Mip Level** and **Go to Previous Mip Level** buttons now appear on the **Image Editor Mode** toolbar. If the **Properties** window is displayed, also notice that the read-only properties **Mip Level** and **Mip Level Count** now appear in the image properties.

Modifying MIP levels

To achieve special effects or increase image quality at specific levels of detail, you can modify each MIP level individually. For example, you can give a textured object a different appearance at a distance (greater distance corresponds to smaller MIP levels), or you can ensure that textures that contain text or symbols remain legible even at smaller MIP levels.

To modify an individual MIP level

1. Select the MIP level that you want to modify. On the **Image Editor Mode** toolbar, use the **Go to Next MIP Level** and **Go to Previous MIP Level** buttons to move between MIP levels.
2. After you select the MIP level that you want to modify, you can use the drawing tools to modify it without changing the contents of other MIP levels. The drawing tools are available on the **Image Editor** toolbar. After you select a tool, you can change its properties in the **Properties** window. For information about the drawing tools and their properties, see [Image Editor](#).

NOTE

If you do not need to modify the contents of individual MIP levels—as you might do to achieve certain effects—we recommend that you generate mipmaps from the source texture at build time. This helps to ensure that MIP levels stay in sync with the source texture because modifications to a MIP level are not propagated to other levels automatically. For more information on how to generate mipmaps at build time, see [How to: Export a texture that contains mipmaps](#).

See also

- [How to: Create a basic texture](#)

Work with 3D models

2/8/2019 • 2 minutes to read • [Edit Online](#)

You can use the Model Editor in Visual Studio to create 3D models. You can use the models in your DirectX-based game or app.

3D models

3D models define the shape of objects as they exist in a 3D scene. Models can be basic solitary objects, complex objects that are formed from hierarchies of basic objects, or even entire 3D scenes. A 3D object is made up of points in 3D space (known as *vertices*), indices that define triangles, lines, or other primitives that are made up of those points, and attributes that might apply on a per-vertex or per-primitive basis—for example, surface normals. Additionally, some information might apply on a per-object basis—for example, which shader and textures will give the object its unique appearance.

The Model Editor is the only tool you need to create basic 3D models—complete with material properties, textures, and pixel shaders—that you can use in your game or app. Or, you can create placeholder models to use for prototyping and testing before you engage artists to finalize the models.

You can also use the Model Editor to view existing 3D models that have been created by using full-featured tools, and to modify them if you observe problems in the art assets.

Related topics

TITLE	DESCRIPTION
Model Editor	Describes how to use the Model Editor to work with 3D models.
Model Editor examples	Provides links to topics that demonstrate how to use the Model Editor to perform common 3D modeling tasks.

Model editor

2/8/2019 • 21 minutes to read • [Edit Online](#)

This document describes how to work with the Visual Studio **Model Editor** to view, create, and modify 3D models.

You can use **Model Editor** to create basic 3D models from scratch, or to view and modify more-complex 3D models that were created by using full-featured 3D modeling tools.

Supported formats

The **Model Editor** supports several 3D model formats that are used in DirectX app development:

FORMAT NAME	FILE EXTENSION	SUPPORTED OPERATIONS (VIEW, EDIT, CREATE)
AutoDesk FBX Interchange File	<i>.fbx</i>	View, Edit, Create
Collada DAE File	<i>.dae</i>	View, Edit (Modifications to Collada DAE files are saved by using the FBX format.)
OBJ	<i>.obj</i>	View, Edit (Modifications to OBJ files are saved by using the FBX format.)

Get started

This section describes how to add a 3D model to your Visual Studio C++ project and other basic information that will help you get started.

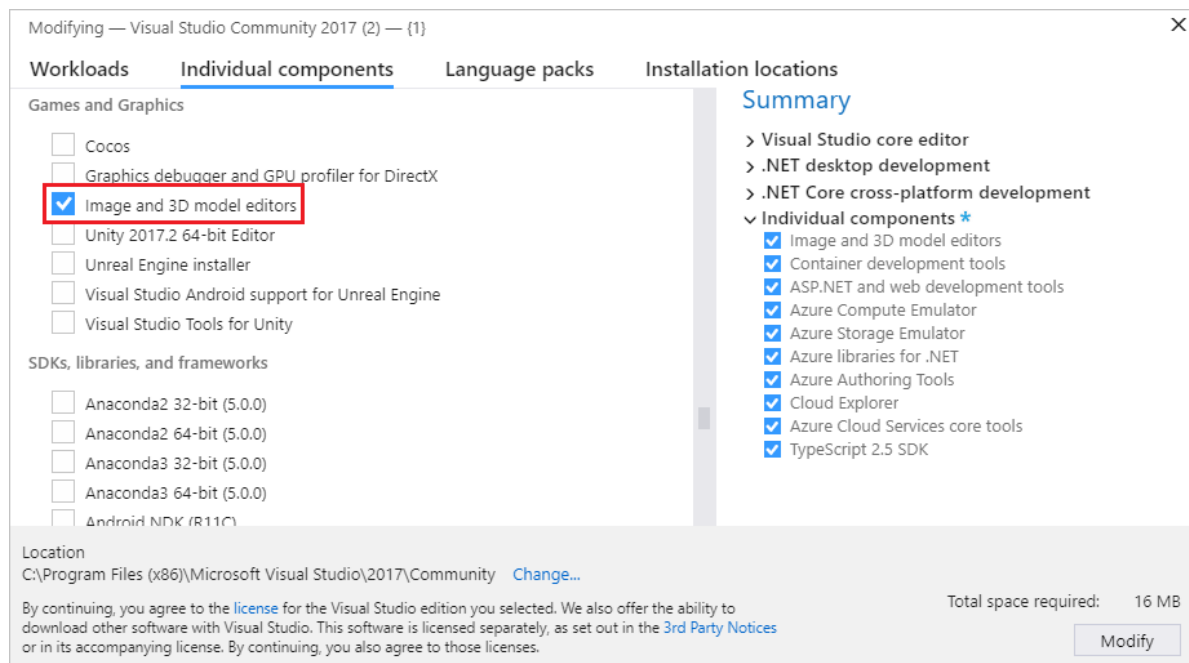
NOTE

Automatic build integration of graphics items like 3D scenes (.fbx files) is only supported for C++ projects.

To add a 3D model to your project

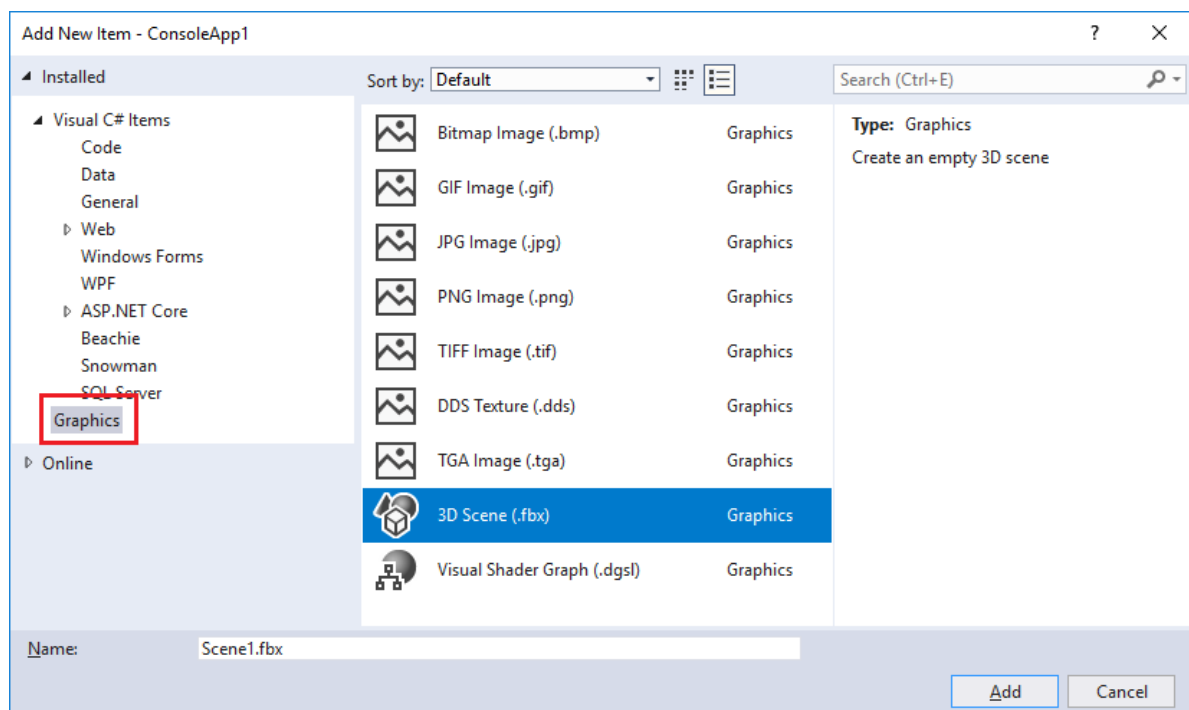
1. Ensure you have the required Visual Studio component installed that you need to work with graphics. The component is called **Image and 3D model editors**.

To install it, open Visual Studio Installer by selecting **Tools > Get Tools and Features** from the menu bar, and then select the **Individual components** tab. Select the **Image and 3D model editors** component under the **Games and Graphics** category, and then select **Modify**.



The component starts installing.

2. In **Solution Explorer**, open the shortcut menu for the C++ project you want to add the image to, and then choose **Add > New Item**.
3. In the **Add New Item** dialog box, under the **Graphics** category, select **3D Scene (.fbx)**.



NOTE

If you don't see the **Graphics** category in the **Add New Item** dialog, and you have the **Image and 3D model editors** component installed, graphics items are not supported for your project type.

4. Enter the **Name** of the model file, and then select **Add**.

Axis orientation

Visual Studio supports every orientation of the 3D axis, and loads axis orientation information from model file formats that support it. If no axis orientation is specified, Visual Studio uses the right-handed coordinate system

by default. The **axis indicator** shows the current axis orientation in the lower-right corner of the design surface. On the **axis indicator**, red represents the x-axis, green represents the y-axis, and blue represents the z-axis.

Begin your 3D model

In the Model Editor, each new object always begins as one of the basic 3D shapes—or *primitives*—that are built into the Model Editor. To create new and unique objects you add a primitive to the scene and then change its shape by modifying its vertices. For complex shapes, you add additional vertices by using extrusion or subdivision and then modify them. For information about how to add a primitive object to your scene, see [Create and import 3D objects](#). For information about how to add more vertices to an object, see [Modify objects](#).

Work with the Model Editor

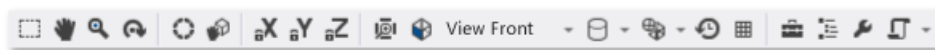
The following sections describe how to use the Model Editor to work with 3D models.

Model Editor toolbars

The Model Editor toolbars contain commands that help you work with 3D models.

Commands that affect the state of the Model Editor are located on the **Model Editor Mode** toolbar in the main Visual Studio window. Modeling tools and scripted commands are located on the **Model Editor** toolbar on the Model Editor design surface.

Here's the **Model Editor Mode** toolbar:



This table describes the items on the **Model Editor Mode** toolbar, which are listed in the order in which they appear from left to right.

TOOLBAR ITEM	DESCRIPTION
Select	Enables selection of points, edges, faces, or objects in the scene, depending on the active selection mode.
Pan	Enables movement of a 3D scene relative to the window frame. To pan, select a point in the scene and move it around. In Select mode, you can press and hold Ctrl to activate Pan mode temporarily.
Zoom	Enables the display of more or less scene detail relative to the window frame. In Zoom mode, select a point in the scene and then move it right or down to zoom in, or left or up to zoom out. In Select mode, you can zoom in or out by using the mouse wheel while you press and hold Ctrl .
Orbit	Positions the view on a circular path around the selected object. If no object is selected, the path is centered on the scene origin. Note: This mode has no effect when Orthographic projection is enabled.
World Local	When this item is enabled, transformations on the selected object occur in world-space. Otherwise, transformations on the selected object occur in local-space.

TOOLBAR ITEM	DESCRIPTION
Pivot Mode	When this item is enabled, transformations affect the location and orientation of the <i>pivot point</i> of the selected object (The pivot point defines the center of translation, scaling, and rotation operations.) Otherwise, transformations affect the location and orientation of the object's geometry, relative to the pivot point.
Lock X axis	Restricts object manipulation to the x axis. Applies only when you use the center part of the manipulator widget.
Lock Y axis	Restricts object manipulation to the y axis. Applies only when you use the center part of the manipulator widget.
Lock Z axis	Restricts object manipulation to the z axis. Applies only when you use the center part of the manipulator widget.
Frame Object	Frames the selected object so that it's located in the center of the view.
View	<p>Sets the view orientation. Here are the available orientations:</p> <p>Front Positions the view in front of the scene.</p> <p>Back Positions the view behind the scene.</p> <p>Left Positions the view to the left of the scene.</p> <p>Right Positions the view to the right of the scene.</p> <p>Top Positions the view above the scene.</p> <p>Bottom Positions the view beneath the scene. Note: This is the only way to change the view direction when Orthographic projection is enabled.</p>
Projection	<p>Sets the kind of projection that is used to draw the scene. Here are the available projections:</p> <p>Perspective In perspective projection, objects that are farther away from the viewpoint appear smaller in size and ultimately converge to a point in the distance.</p> <p>Orthographic In Orthographic projection, objects appear to be the same size, regardless of their distance from the viewpoint. No convergence is displayed. When Orthographic projection enabled, you can't use Orbit mode to position the view.</p>

TOOLBAR ITEM	DESCRIPTION
Draw Style	<p>Sets how objects in the scene are rendered. Here are the available styles:</p> <p>Wire Frame When enabled, objects are rendered as wireframes.</p> <p>Overdraw When enabled, objects are rendered by using additive blending. You can use this to visualize how much overdraw is occurring in the scene.</p> <p>Flat Shaded When enabled, objects are rendered by using a basic, flat shaded lighting model. You can use this to see the faces of an object more easily.</p> <p>If none of these options are enabled, each object is rendered by using the material that's applied to it.</p>
Real-Time Rendering Mode	<p>When real-time rendering is enabled, Visual Studio redraws the design surface, even when no user action is performed. This mode is useful when you work with shaders that change over time.</p>
Toggle Grid	<p>When this item is enabled, a grid is displayed. Otherwise, the grid is not displayed.</p>
Toolbox	<p>Alternately shows or hides the Toolbox.</p>
Document Outline	<p>Alternately shows or hides the Document Outline window.</p>
Properties	<p>Alternately shows or hides the Properties window.</p>
Advanced	<p>Contains advanced commands and options.</p> <p>Graphics Engines</p> <p>Render with D3D11 Uses Direct3D 11 to render the Model Editor design surface.</p> <p>Render with D3D11WARP Uses Direct3D 11 Windows Advanced Rasterization Platform (WARP) to render the Model Editor design surface.</p> <p>Scene Management</p> <p>Import Imports objects from another 3D model file to the current scene.</p> <p>Attach to Parent Establishes the first of multiple selected objects as the parent of the remaining selected objects.</p> <p>Detach from Parent Detaches the selected object from its parent. The selected object becomes a <i>root object</i> in the scene. A root object doesn't have a parent object.</p>

TOOLBAR ITEM	Create Group DESCRIPTION Groups the selected objects as sibling objects.
	<p>Merge Objects Combines the selected objects into one object.</p> <p>Create New Object From Polygon Selection Removes the selected faces from the current object and adds to the scene a new object that contains those faces.</p> <p>Tools</p> <p>Flip Polygon Winding Flips the selected polygons so that its winding order and surface normal are inverted.</p> <p>Remove All Animation Removes animation data from the objects.</p> <p>Triangulate Converts the selected object to triangles.</p> <p>View</p> <p>Backface Culling Enables or disables backface culling.</p> <p>Frame Rate Displays the frame rate in the upper-right corner of the design surface. The frame rate is the number of frames that are drawn per second.</p> <p>This option is useful when you enable the Real-Time Rendering Mode option.</p> <p>Show All Shows all objects in the scene. This resets the Hidden property of each object to False.</p> <p>Show Face Normals Shows the normal of each face.</p> <p>Show Missing Materials Displays a special texture on objects that don't have a material assigned to them.</p> <p>Show Pivot Enables or disables the display of a 3D axis marker at the pivot point of the active selection.</p> <p>Show Placeholder Nodes Shows placeholder nodes. A placeholder node is created when you group objects.</p> <p>Show Vertex Normals Shows the normal of each vertex. Tip: You can choose the Scripts button to run the last script again.</p>

Here's the **Model Editor** toolbar:



The next table describes the items on the **Model Editor** toolbar, which are listed in the order in which they appear from top to bottom.

TOOLBAR ITEM	DESCRIPTION
Translate	Moves the selection.
Scale	Changes the size of the selection.
Rotate	Rotates the selection.
Select Point	Sets the Selection mode to select individual points on an object.
Select Edge	Sets the Selection mode to select an edge (a line between two vertices) on an object.
Select Face	Sets the Selection mode to select a face on an object.
Select Object	Sets the Selection mode to select an entire object.
Extrude	Creates an additional face and connects it to the selected face.
Subdivide	Divides each selected face into multiple faces. To create the new faces, new vertices are added—one in the center of the original face, and one in the middle of each edge—and then joined together with the original vertices. The number of added faces is equal to the number of edges in the original face.

Control the view

The 3D scene is rendered according to the view, which can be thought of as a virtual camera that has a position and an orientation. To change the position and orientation, use the view controls on the **Model Editor Mode** toolbar.

The following table describes the primary view controls.

VIEW CONTROL	DESCRIPTION
Pan	<p>Enables movement of a 3D scene relative to the window frame. To pan, select a point in the scene and move it around.</p> <p>In Select mode, you can press and hold Ctrl to activate Pan mode temporarily.</p>

VIEW CONTROL	DESCRIPTION
Zoom	<p>Enables the display of more or less scene detail relative to the window frame. In Zoom mode, select a point in the scene and then move it right or down to zoom in, or left or up to zoom out.</p> <p>In Select mode, you can zoom in or out by using the mouse wheel while you press and hold Ctrl.</p>
Orbit	<p>Positions the view on a circular path around the selected object. If no object is selected, the path is centered on the scene origin. Note: This mode has no effect when Orthographic projection is enabled.</p>
Frame Object	<p>Frames the selected object so that it's located in the center of the view.</p>

The view is established by the virtual camera, but it's also defined by a projection. The projection defines how shapes and objects in the view are translated into pixels on the design surface. On the **Model Editor** toolbar, you can choose either **Perspective** or **Orthographic** projection.

PROJECTION	DESCRIPTION
Perspective	<p>In perspective projection, objects that are farther away from the viewpoint appear smaller in size and ultimately converge to a point in the distance.</p>
Orthographic	<p>In Orthographic projection, objects appear to be the same size, regardless of their distance from the viewpoint. No convergence is displayed. When Orthographic projection enabled, you can't use Orbit mode to position the view arbitrarily.</p>

You might find it useful to view a 3D scene from a known position and angle, for example, when you want to compare two similar scenes. For this scenario, the Model Editor provides several predefined views. To use a predefined view, on the **Model Editor Mode** toolbar, choose **View**, and then choose the predefined view you want—front, back, left, right, top, or bottom. In these views, the virtual camera looks directly at the origin of the scene. For example, if you choose **View Top**, the virtual camera looks at the origin of the scene from directly above it.

View additional geometry details

To better understand a 3D object or scene, you can view additional geometry details such as per-vertex normals, per-face normals, the pivot points of the active selection, and other details. To enable or disable them, on the **Model Editor** toolbar, choose **Scripts > View**, and then choose the one you want.

Create and import 3D objects

To add a predefined 3D shape to the scene, in the **Toolbox**, select the one you want and then move it to the design surface. New shapes are positioned at the origin of the scene. The Model Editor provides seven shapes: **Cone**, **Cube**, **Cylinder**, **Disc**, **Plane**, **Sphere**, and **Teapot**.

To import a 3D object from a file, on the **Model Editor** toolbar, choose **Advanced > Scene Management > Import >** and then specify the file that you want to import.

Transform objects

You can *transform* an object by changing its **Rotation**, **Scale**, and **Translation** properties. *Rotation* orients an

object by applying successive rotations around the x-axis, y-axis, and z-axis defined by its pivot point. Each rotation specification has three components—x, y, and z, in that order—and the components are specified in degrees. **Scaling** resizes an object by stretching it by a specified factor along one or more axes centered on its pivot point. *Translation* locates an object in 3-dimensional space relative to its parent instead of its pivot point.

You can transform an object either by using modeling tools or by setting properties.

Transform an object by using modeling tools

1. In **Select** mode, select the object you want to transform. A wireframe overlay indicates that the object is selected.
2. On the **Model Editor** toolbar, choose the **Translate**, **Scale**, or **Rotate** tool. A translation, scaling, or rotation manipulator appears for the selected object.
3. Use the manipulator to perform the transformation. For translation and scaling transformations, the manipulator is an axis indicator. You can change one axis at a time, or you can change all axes at the same time by using the white cube at the center of the indicator. For rotation, the manipulator is a sphere made of color-coded circles that correspond to the x-axis (red), y-axis (green), and z-axis (blue). You have to change each axis individually to create the rotation you want.

Transform an object by setting its properties

1. In **Select** mode, select the object that you want to transform. A wireframe overlay indicates that the object is selected.
2. In the **Properties** window, specify values for the **Rotation**, **Scale**, and **Translation** properties.

IMPORTANT

For the **Rotation** property, specify the degree of rotation around each of the three axes. Rotations are applied in order, so make sure to plan a rotation, first in terms of the x-axis rotation, then the y-axis, and then the z-axis.

By using the modeling tools, you can create transformations quickly but not precisely. By setting the object properties, you can specify transformations precisely but not quickly. We recommend that you use the modeling tools to get "close enough" to the transformations you want, and then fine-tune the property values.

If you don't want to use manipulators, you can enable free-form mode. On the **Model Editor** toolbar, choose **Scripts > Tools > Free-form Manipulation** to enable (or disable) free-form mode. In free-form mode, you can begin a manipulation at any point on the design surface instead of a point on the manipulator. In free-form mode, you can constrain changes to certain axes by locking the ones you don't want to change. On the **Model Editor Mode** toolbar, choose any combination of the **Lock X**, **Lock Y**, and **Lock Z** buttons.

You might find it useful to work with objects by using snap-to-grid. On the **Model Editor Mode** toolbar, choose **Snap** to enable (or disable) snap-to-grid. When snap-to-grid is enabled, translation, rotation, and scaling transformations are constrained to predefined increments.

Work with the pivot point

The pivot point of an object defines its center of rotation and scaling. You can change the pivot point of an object to change how it's affected by rotation and scaling transformations. On the **Model Editor Mode** toolbar, choose **Pivot Mode** to enable (or disable) pivot mode. When pivot mode is enabled, a small axis indicator appears at the pivot point of the selected object. You can then use the **Translation** and **Rotation** tools to manipulate the pivot point.

For a demonstration that shows how to use the pivot point, see [How to: Modify the pivot point of a 3D model](#).

World and local modes

Translation and rotation can occur in either the local coordinate system (or *local frame-of-reference*) of the object, or in the coordinate system of the world (or the *world frame-of-reference*). The world frame-of-reference is

independent of the rotation of the object. Local mode is the default. To enable (or disable) world mode, on the **Model Editor Mode** toolbar, choose the **WorldLocal** button.

Modify objects

You can change the shape of a 3D object by moving or deleting its vertices, edges, and faces. By default, the Model Editor is in *object mode*, so that you can select and transform entire objects. To select points, edges, or faces, choose the appropriate selection mode. On the **Model Editor Mode** toolbar, choose **Selection modes**, and then choose the mode that you want.

You can create additional vertices by extrusion or by subdivision. Extrusion duplicates the vertices of a face (a coplanar set of vertices), which remain connected by the duplicated vertices. Subdivision adds vertices to create several faces where there was previously one. To create the new faces, new vertices are added—one in the center of the original face, and one in the middle of each edge—and then joined together with the original vertices. The number of added faces is equal to the number of edges in the original face. In both cases, you can translate, rotate, and scale the new vertices to change the geometry of the object.

To extrude a face from an object

1. In face-select mode, select the face you want to extrude.
2. On the **Model Editor** toolbar, choose **Scripts** > **Tools** > **Extrude**.

To subdivide faces

1. In face-select mode, select the faces you want to subdivide. Because subdivision creates new edge data, subdividing all faces at once gives more-consistent results when the faces are adjacent.
2. On the **Model Editor** toolbar, choose **Scripts** > **Tools** > **Subdivide**.

You can also triangulate faces, merge objects, and convert polygon selections into new objects. Triangulation creates additional edges such that a non-triangular face is converted to an optimal number of triangles; however, it doesn't provide additional geometric detail. Merging combines selected objects into one object. New objects can be created from a polygon selection.

Triangulate a face

1. In face-select mode, select the face you want to triangulate.
2. On the **Model Editor** toolbar, choose **Scripts** > **Tools** > **Triangulate**.

Merge objects

1. In object-select mode, select the objects you want to merge.
2. On the **Model Editor** toolbar, choose **Scripts** > **Tools** > **Merge Objects**.

Create an object from a polygon selection

1. In face-select mode, select the faces you want to create a new object from.
2. On the **Model Editor** toolbar, choose **Scripts** > **Tools** > **Create New Object from Polygon Selection**.

Work with materials and shaders

The appearance of an object is determined by the interaction of lighting in the scene and the material of the object. Materials are defined by properties that describe how the surface reacts to different types of light and by a shader program that calculates the final color of each pixel on the object surface based on lighting information, texture maps, normal maps, and other data.

The Model Editor provides these default materials:

MATERIAL	DESCRIPTION
Unlit	Renders a surface without any simulated lighting.

MATERIAL	DESCRIPTION
Lambert	Renders a surface with simulated ambient lighting and diffuse lighting.
Phong	Renders a surface with simulated ambient lighting, diffuse lighting, and specular highlights.

Each of these materials applies one texture on the surface of an object. You can set a different texture for each object that uses the material.

To modify how a particular object reacts to the different light sources in the scene, you can change the lighting properties of material independent of other objects that use the material. This table describes common lighting properties:

LIGHTING PROPERTY	DESCRIPTION
Ambient	Describes how the surface is affected by ambient lighting.
Diffuse	Describes how the surface is affected by directional and point lights.
Emissive	Describes how the surface emits light, independent of other lighting.
Specular	Describes how the surface reflects directional and point lights.
Specular Power	Describes the breadth and intensity of specular highlights.

Depending on what a material supports, you can change its lighting properties, textures, and other data. In **Select** mode, select the object whose material you want to change, and then in the **Properties** window, change the **MaterialAmbient**, **MaterialDiffuse**, **MaterialEmissive**, **MaterialSpecular**, **MaterialSpecularPower**, or other available property. A material can expose up to eight textures, whose properties are named sequentially from **Texture1** to **Texture8**.

To remove all materials from an object, on the **Model Editor** toolbar, choose **Scripts > Materials > Remove Materials**.

You can use the **Shader Designer** to create custom shader materials that you can apply to objects in your 3D scene. For information about how to create custom shader materials, see [Shader Designer](#). For information about how to apply a custom shader material to an object, see [How to: Apply a shader to a 3D model](#).

Scene management

You can manage scenes as a hierarchy of objects. When multiple objects are arranged in a hierarchy, any translation, scale, or rotation of a parent node also affects its children. This is useful when you want to construct complex objects or scenes from more basic objects.

You can use the **Document Outline** window to view the scene hierarchy and select scene nodes. When you select a node in the outline, you can use the **Properties** window to modify its properties.

You can construct a hierarchy of objects either by making one of them the parent to the others or by grouping them together as siblings under a placeholder node that acts as the parent.

Create a hierarchy that has a parent object

1. In **Select** mode, select two or more objects. The first one you select will be the parent object.

2. On the **Model Editor** toolbar, choose **Scripts > Scene Management > Attach to Parent**.

Create a hierarchy of sibling objects

1. In **Select** mode, select two or more objects. A placeholder object is created and becomes their parent object.
2. On the **Model Editor** toolbar, choose **Scripts > Scene Management > Create Group**.

The Model Editor uses a white wireframe to identify the first selected object, which becomes the parent. Other objects in the selection have a blue wireframe. By default, placeholder nodes are not displayed. To display placeholder nodes, on the **Model Editor** toolbar, choose **Scripts > Scene Management > Show Placeholder Nodes**. You can work with placeholder nodes just as you work with non-placeholder objects.

To remove the parent-child association between two objects, select the child object, and then on the **Model Editor** toolbar, choose **Scripts > Scene Management > Detach from Parent**. When you detach the parent from a child object, the child object becomes a root object in the scene.

Keyboard shortcuts

COMMAND	KEYBOARD SHORTCUTS
Switch to Select mode	Ctrl+G, Ctrl+Q S
Switch to Zoom mode	Ctrl+G, Ctrl+Z Z
Switch to Pan mode	Ctrl+G, Ctrl+P K
Select all	Ctrl+A
Delete the current selection	Delete
Cancel the current selection	Escape (Esc)
Zoom in	Mouse wheel forward Ctrl+Mouse wheel forward Shift+Mouse wheel forward Ctrl+PageUp Plus Sign (+)
Zoom out	Mouse wheel backward Ctrl+Mouse wheel backward Shift+Mouse wheel backward Ctrl+PageDown Minus Sign (-)

COMMAND	KEYBOARD SHORTCUTS
Pan the camera up	PageDown
Pan the camera down	PageUp
Pan the camera left	Mouse wheel left Ctrl+PageDown
Pan the camera right	Mouse wheel right Ctrl+PageDown
View top of model	Ctrl+L, Ctrl+T T
View bottom of model	Ctrl+L, Ctrl+U
View left side of model	Ctrl+L, Ctrl+L
View right side of model	Ctrl+L, Ctrl+R
View front of model	Ctrl+L, Ctrl+F
View back of model	Ctrl+L, Ctrl+B
Frame object in window	F
Toggle wireframe mode	Ctrl+L, Ctrl+W
Toggle snap-to-grid	Ctrl+G, Ctrl+N
Toggle pivot mode	Ctrl+G, Ctrl+V
Toggle x-axis restriction	Ctrl+L, Ctrl+X
Toggle y-axis restriction	Ctrl+L, Ctrl+Y
Toggle z-axis restriction	Ctrl+L, Ctrl+Z
Switch to translation mode	Ctrl+G, Ctrl+W W
Switch to scale mode	Ctrl+G, Ctrl+E E
Switch to rotation mode	Ctrl+G, Ctrl+R R

COMMAND	KEYBOARD SHORTCUTS
Switch to point-select mode	Ctrl+L, Ctrl+1
Switch to edge-select mode	Ctrl+L, Ctrl+2
Switch to face-select mode	Ctrl+L, Ctrl+3
Switch to object-select mode	Ctrl+L, Ctrl+4
Switch to orbit (camera) mode	Ctrl+G, Ctrl+O
Select next object in scene	Tab
Select previous object in scene	Shift+Tab
Manipulate the selected object based on the current tool.	The Arrow keys
Deactivate current manipulator	Q
Rotate camera	Alt+ Drag with left mouse button

Related topics

TITLE	DESCRIPTION
Working with 3D assets for games and apps	Provides an overview of the Visual Studio tools that you can use to work with graphics assets such as textures and images, 3D models, and shader effects.
Image Editor	Describes how to use the Visual Studio Image Editor to work with textures and images.
Shader Designer	Describes how to use the Visual Studio Shader Designer to work with shaders.

Model Editor examples

2/8/2019 • 2 minutes to read • [Edit Online](#)

The articles in this section of the documentation contain examples that demonstrate how you can use the Model Editor.

Related topics

TITLE	DESCRIPTION
How to: Create a basic 3D model	Describes how to create a basic 3D model.
How to: Modify the pivot point of a 3D model	Describes how to modify the pivot point and scale of a 3D model.
How to: Model 3D terrain	Describes how to create a basic landscape scene.
How to: Apply a shader to a 3D model	Describes how to apply a shader to a 3D model.

How to: Create a basic 3D model

2/8/2019 • 2 minutes to read • [Edit Online](#)

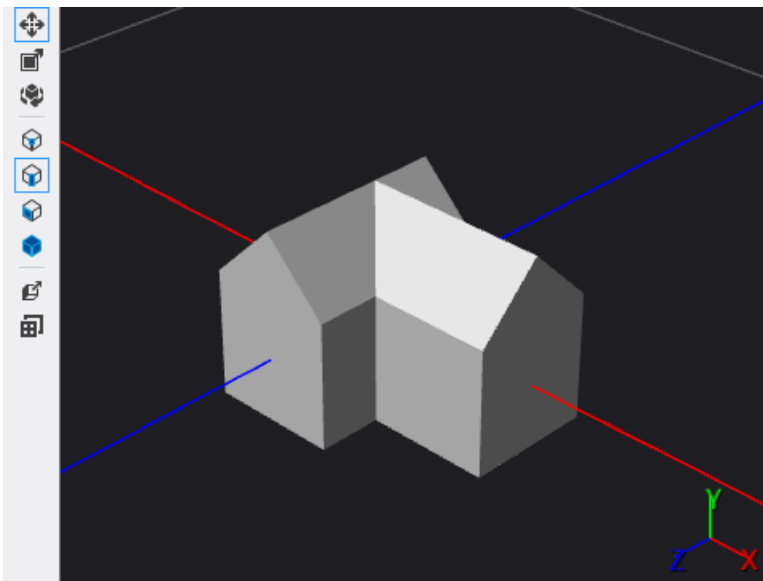
This article demonstrates how to use the Model Editor to create a basic 3D model. The following activities are covered:

- Adding objects to a scene
- Selecting faces and edges
- Translating selections
- Using the **Subdivide face** and **Extrude face** tools
- Using the **Triangulate** command

Create a basic 3D model

You can use the Model Editor to create and modify 3D models and scenes for your game or app. The following steps show how to use the Model Editor to create a simplified 3D model of a house. A simplified model can be used as a stand-in for final art assets that are still being created, as a mesh for collision detection, or as a low-detail model to be used when the object that it represents is too far away to benefit from more detailed rendering.

When you're finished, the model should look like this:

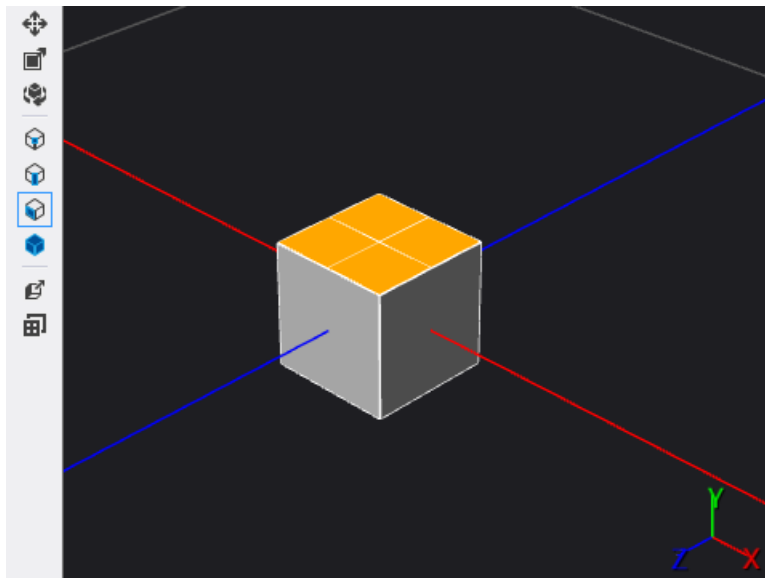


Before you begin, make sure that the **Properties** window and **Toolbox** are displayed.

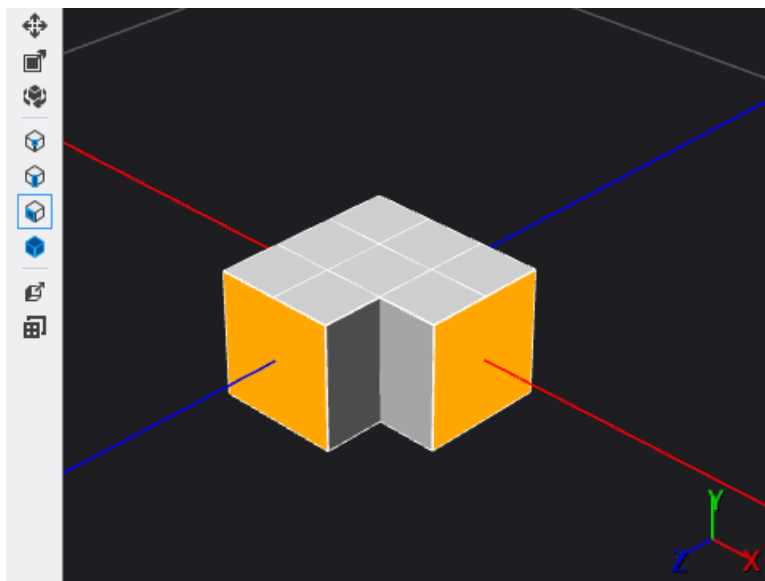
To create a simplified 3D model of a house

1. Create a 3D model with which to work. For information about how to add a model to your project, see the Getting Started section in [Model Editor](#).
2. Add a cube to the scene. In the **Toolbox** window, under **Shapes**, select **Cube** and then move it to the design surface.
3. Switch to face-selection. On the Model Editor toolbar, choose **Select Face**.
4. Subdivide the top of the cube. In face selection mode, choose the cube once to activate it for selection, and then choose the top of the cube to select the top face. On the Model Editor toolbar, choose **Subdivide face**.

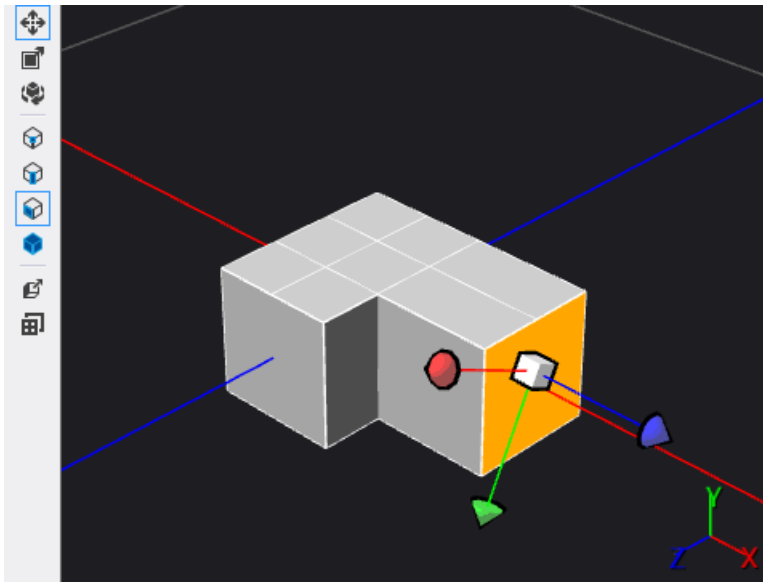
This adds new vertices to the top of the cube that split it into four equally sized partitions.



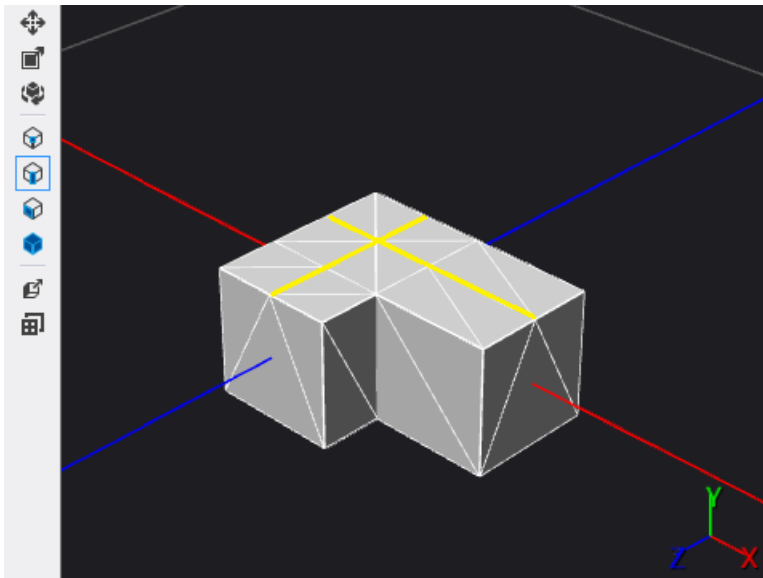
5. Extrude two adjacent sides of the cube—for example, the front and right sides of the cube. In face selection mode, choose the cube once to activate it for selection and then choose one side of the cube. Press and hold the **Ctrl** key, choose another side of the cube that is adjacent to the side you selected first, and then on the Model Editor toolbar, choose **Extrude face**.



6. Extend one of the extrusions. Choose one of the faces that you just extruded, and then, on the Model Editor toolbar, choose the **Translate** tool and move the translation manipulator in the same direction as the extrusion.

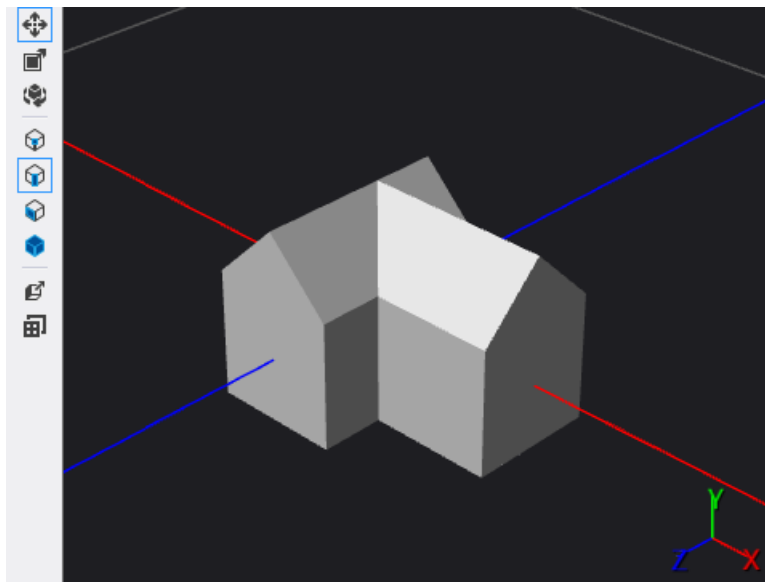


7. Triangulate the model. On the Model Editor toolbar, choose **Advanced > Tools > Triangulate**.
8. Create the roof of the house. Switch to edge-selection mode by choosing **Select Edge** on the Model Editor toolbar, and then choose the cube to activate it. Press and hold the **Ctrl** key as you select the edges that are shown here:



When the edges are selected, on the Model Editor toolbar, choose the **Translate** tool and then move the translation manipulator upward to create the roof of the house.

The simplified house model is complete. Here's the final model again, with flat shading applied:



As a next step, you can apply a shader to this 3D model. For information, see [How to: Apply a shader to a 3D model](#).

See also

- [How to: Model 3D terrain](#)
- [Model editor](#)
- [Shader designer](#)

How to: Modify the pivot point of a 3D model

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Model Editor to modify the *pivot point* of a 3D model. The pivot point is the point in space that defines the mathematical center of the object for rotation and scaling.

Modify the pivot point of a 3D model

You can redefine the origin of a 3D model by modifying its pivot point.

Make sure that the **Properties** window and the **Toolbox** are displayed.

1. Begin with an existing 3D model, such as the one that's described in [How to: Create a basic 3D model](#).
2. Enter pivot mode. On the **Model Editor Mode** toolbar, choose the **Pivot Mode** button to activate pivot mode. A box appears around the **Pivot Mode** button to indicate that the Model Editor is now in pivot mode. In pivot mode, operations such as translation affect the pivot point of the object instead of the structure of the object in world-space.
3. Modify the pivot point of the object. In **Select** mode, select the object, and then on the **Model Viewer** toolbar, choose the **Translate** tool. A box that represents the pivot point appears on the design surface. Move the box to modify the pivot point of the object.

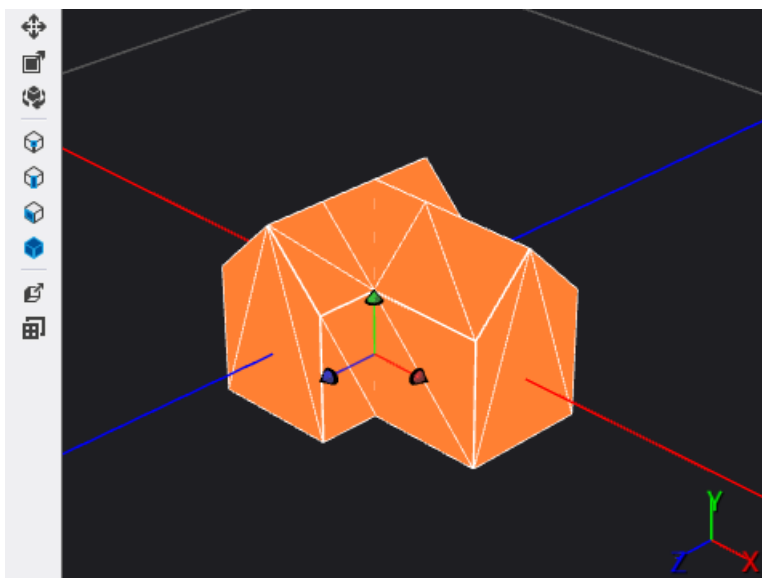
By moving the box, you can move the pivot point in all three dimensions. To translate the pivot point along one axis, move the arrow that corresponds to that axis. The box and arrows change to a yellow color to indicate the axis that's affected by the translation.

You can also specify the pivot point by using the **Pivot Translation** property in the **Properties** window.

TIP

You can view the effect of the new pivot point by rotating the object. To rotate it, use the **Rotate** tool or modify the **Rotation** property.

Here's a model that has a modified pivot point:



See also

- [How to: Create a basic 3D model](#)
- [Model editor](#)

How to: Model 3D terrain

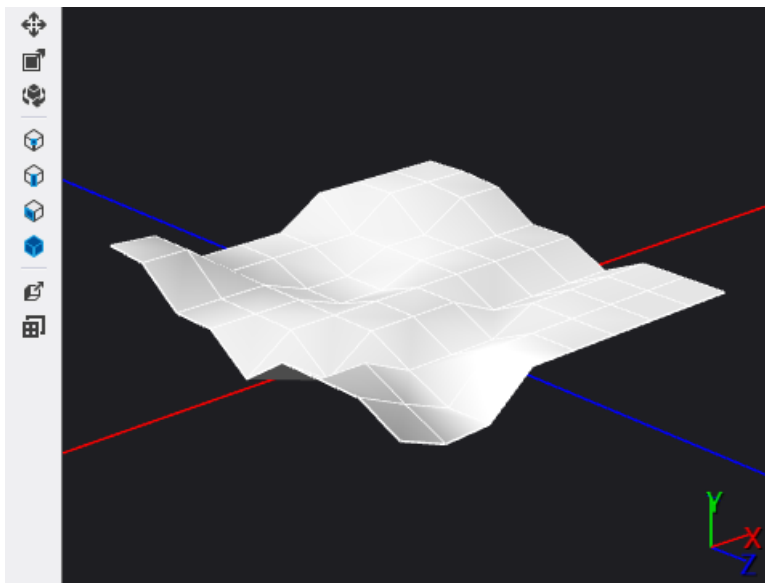
2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Model Editor to create a 3D terrain model.

Create a 3D terrain model

You can create a 3D terrain by subdividing a plane to make additional faces, and then manipulating their vertices to create interesting terrain features.

When you're finished, the model should look like this:



Before you begin, make sure that the **Properties** window and **Toolbox** are displayed.

1. Create a 3D model with which to work. For information about how to add a model to your project, see the Getting Started section in [Model editor](#).
2. Add a plane to the scene. In the **Toolbox**, under **Shapes**, select **Plane** and move it to the design surface.

TIP

To make the plane object easier to work with, you can frame it in the design surface. In **Select** mode, select the plane object, and then on the Model Editor toolbar, choose the **Frame Object** button.

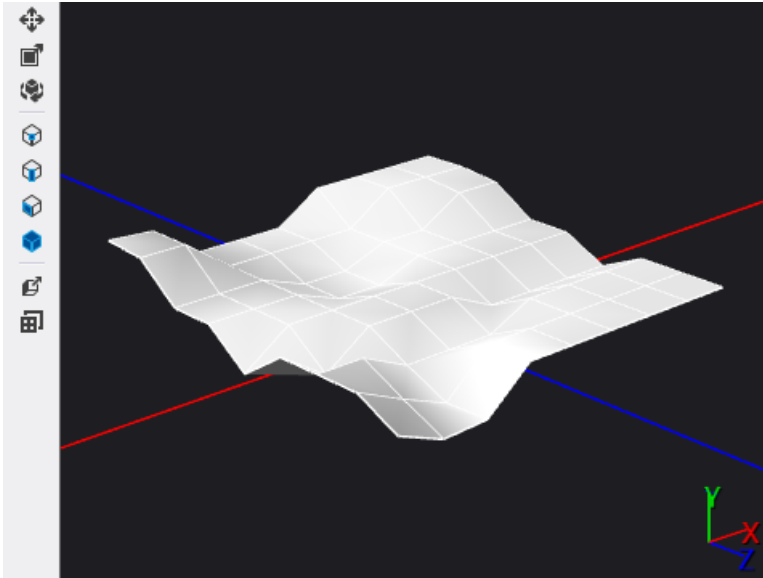
3. Enter face selection mode. On the Model Editor toolbar, choose **Select Face**.
4. Subdivide the plane. In face selection mode, choose the plane once to activate it for selection, and then choose it again to select its only face. On the Model Editor toolbar, choose **Subdivide face**. This adds new vertices to the plane that split it into four equally sized partitions.
5. Create more subdivisions. With the new faces still selected, choose **Subdivide face** two more times. This creates a total of 64 faces. By creating more subdivisions, you can give the terrain even more detail.
6. Enter point selection mode. On the Model Editor toolbar, choose **Select Point**.
7. Modify a point to create a terrain feature. In point selection mode, select one of the points, and then on the Model Editor toolbar, choose the **Translate** tool. A box that represents the point appears on the design

surface. Use the green arrow to move the box and thereby modify the height of the point. Repeat this step for different points to create interesting terrain features.

TIP

You can select several points at once to modify them in a uniform manner.

The terrain model is complete. Here's the final model again, with Phong shading applied:



You can use this terrain model to demonstrate the effect of the gradient shader that's described in [How to: Create a geometry-based gradient shader](#).

See also

- [Model editor](#)

How to: Apply a shader to a 3D model

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Model Editor to apply a Directed Graph Shader Language (DGSL) shader to a 3D model.

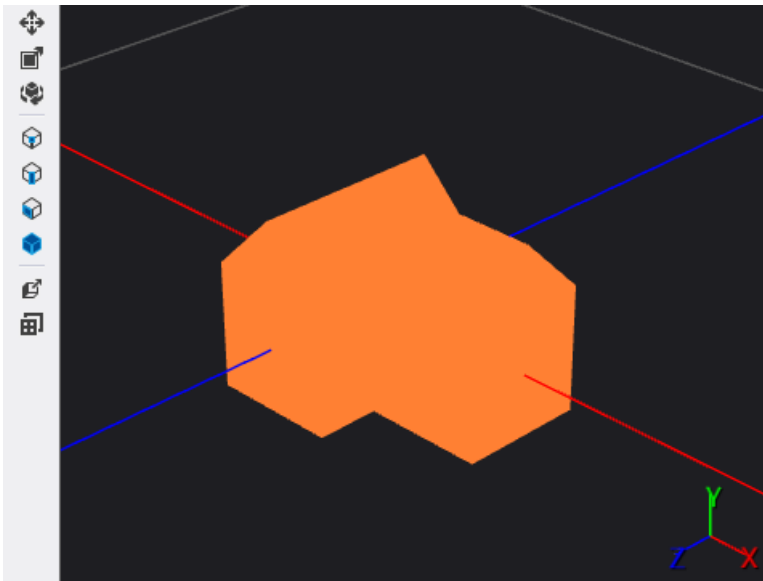
Apply a shader to a 3D model

You can apply a shader effect to a 3D model to give it an interesting appearance.

Before you begin, make sure that the **Properties** window is displayed.

1. Begin with a 3D scene that contains one or more models. If you don't have a suitable 3D scene, create one as described in [How to: Create a basic 3D Model](#). You must also have a DGSL shader that you can apply to the model. If you don't have a suitable shader, create one as described in [How to: Create a basic color shader](#) and make sure that you've saved it to a file before you continue.
2. In **Select** mode, select the model to which you want to apply the shader, and then in the **Properties** window, in the **Filename** property of the **Effect** property group, specify the DGSL shader that you want to apply to the model.

Here's a model that has the basic color effect applied to it:



After you apply a shader to a model, you can open it in the Shader Designer by selecting the model, and then in the **Properties** window, in the **(Advanced)** property of the **Effect** property group, choosing the ellipsis (...) button.

See also

- [How to: Create a basic 3D model](#)
- [How to: Create a basic color shader](#)
- [Model editor](#)
- [Shader designer](#)

Work with shaders

2/8/2019 • 2 minutes to read • [Edit Online](#)

You can use the graph-based Shader Designer in Visual Studio to design custom shader effects. You can use these shaders in your DirectX-based game or app.

Shaders

A *shader* is a computer program that performs graphics calculations—for example, vertex transformations or pixel coloring—and typically runs on a graphics processing unit (GPU) instead of the CPU. Because most stages of the traditional, fixed-function graphics pipeline are now performed by shader programs, you can use them to create a pipeline that's specific to the needs of your app.

The most common kinds of shaders are *vertex shaders*, which perform per-vertex calculations and replace the fixed-function transformation and lighting circuitry in non-programmable graphics hardware, and *pixel shaders*, which perform per-pixel calculations that determine the color of a pixel and replace the fixed-function color-combiner circuitry in non-programmable graphics hardware. Modern graphics hardware has made even more kinds of shaders possible—*hull shaders*, *domain shaders*, and *geometry shaders* for graphics calculations, and *compute shaders* for non-graphics computations. None of these stages are even available in non-programmable graphics hardware. Shaders were originally created by using an assembly-like language that provided data-parallel (SIMD) and graphics-centric (dot product) instructions. Now, shaders are typically created by using high-level, C-like languages like HLSL (High Level Shader Language).

You can use the Shader Designer to create pixel shaders interactively instead of by entering and compiling code. In the Shader Designer, a shader is defined by a number of nodes that represent data and operations, and connections between nodes that represent the flow of data values and intermediate results through the shader. By using this approach and the real-time preview in the Shader Designer, you can visualize the execution of the shader more easily, and "discover" interesting shader variations through experimentation.

DGSL documents

The Shader Designer saves shaders in the Directed Graph Shader Language (DGSL) format, which is an XML format that's based on Directed Graph Markup Language (DGML). You can apply DGSL shaders directly to 3D models in the Model Editor. However, before you can use a DGSL shader in your app, you must export it to a format that DirectX understands—for example, HLSL.

Because DGSL is compatible with DGML, you can use tools that are designed to analyze DGML documents to analyze your DGSL shaders. For information about DGML, see [Understanding Directed Graph Markup Language \(DGML\)](#).

Related topics

TITLE	DESCRIPTION
Shader Designer	Describes how to use the Visual Studio Shader Designer to work with shaders.
Shader Designer nodes	Discusses the kinds of Shader Designer nodes that you can use to achieve graphics effects.

TITLE	DESCRIPTION
Shader Designer examples	Provides links to topics that demonstrate how to use the Shader Designer to achieve common graphics effects.

Shader Designer

2/8/2019 • 10 minutes to read • [Edit Online](#)

This document describes how to work with the Visual Studio **Shader Designer** to create, modify, and export custom visual effects that are known as *shaders*.

You can use **Shader Designer** to create custom visual effects for your game or app even if you don't know high-level shader language (HLSL) programming. To create a shader in **Shader Designer**, you lay it out as a graph. That is, you add to the design surface *nodes* that represent data and operations and then make connections between them to define how the operations process the data. At each operation node, a preview of the effect up to that point is provided so that you can visualize its result. Data flows through the nodes toward a final node that represents the output of the shader.

Supported formats

The **Shader Designer** supports these shader formats:

FORMAT NAME	FILE EXTENSION	SUPPORTED OPERATIONS (VIEW, EDIT, EXPORT)
Directed Graph Shader Language	.dgsi	View, Edit
HLSL Shader (source code)	.hlsl	Export
HLSL Shader (bytecode)	.cso	Export
C++ header (HLSL bytecode array)	.h	Export

Get started

This section describes how to add a DGSL shader to your Visual Studio C++ project and provides basic information to help you get started.

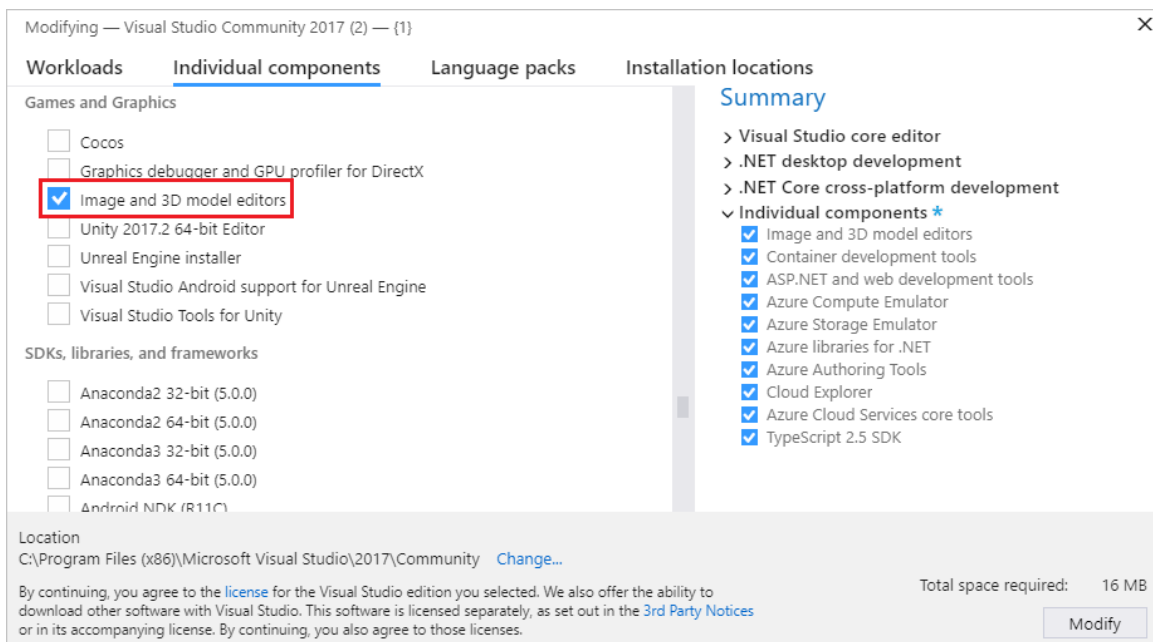
NOTE

Automatic build integration of graphics items like shader graphs (.dgsi files) is only supported for C++ projects.

To add a DGSL shader to your project

1. Ensure you have the required Visual Studio component installed that you need to work with graphics. The component is called **Image and 3D model editors**.

To install it, open Visual Studio Installer by selecting **Tools > Get Tools and Features** from the menu bar, and then select the **Individual components** tab. Select the **Image and 3D model editors** component under the **Games and Graphics** category, and then select **Modify**.



2. In **Solution Explorer**, open the shortcut menu for the C++ project to which you want to add the shader, and then choose **Add > New Item**.
3. In the **Add New Item** dialog box, under **Installed**, select **Graphics**, and then select **Visual Shader Graph (.dgsi)**.

NOTE

If you don't see the **Graphics** category in the **Add New Item** dialog, and you have the **Image and 3D model editors** component installed, graphics items are not supported for your project type.

4. Specify the **Name** of the shader file, and the **Location** where you want it to be created.
5. Choose the **Add** button.

The default shader

Each time that you create a DGSL shader, it begins as a minimal shader that has just a **Point Color** node that's connected to the **Final Color** node. Although this shader is complete and functional, it doesn't do much. Therefore, the first step in creating a working shader is often to delete the **Point Color** node or disconnect it from the **Final Color** node to make room for other nodes.

Work with the Shader Designer

The following sections describe how to use the Shader Designer to work with custom shaders.

Shader Designer toolbars

The Shader Designer toolbars contain commands that help you work with DGSL shader graphs.

Commands that affect the state of the Shader Designer are located on the **Shader Designer Mode** toolbar in the main Visual Studio window. Design tools and commands are located on the **Shader Designer** toolbar on the Shader Designer design surface.

Here's the **Shader Designer Mode** toolbar:



This table describes the items on the **Shader Designer Mode** toolbar, which are listed in the order in which they appear from left to right:

TOOLBAR ITEM	DESCRIPTION
Select	Enables interaction with nodes and edges in the graph. In this mode, you can select nodes and move or delete them, and you can establish edges or break them.
Pan	<p>Enables movement of a shader graph relative to the window frame. To pan, select a point on the design surface and move it around.</p> <p>In Select mode, you can press and hold Ctrl to activate Pan mode temporarily.</p>
Zoom	<p>Enables the display of more or less shader-graph detail relative to the window frame. In Zoom mode, select a point on the design surface and then move it right or down to zoom in, or left or up to zoom out.</p> <p>In Select mode, you can press and hold Ctrl to zoom in or out by using the mouse wheel.</p>
Zoom to Fit	Displays the full shader graph in the window frame.
Real-Time Rendering Mode	When real-time rendering is enabled, Visual Studio redraws the design surface, even when no user action is performed. This mode is useful when you work with shaders that change over time.
Preview with sphere	When enabled, a model of a sphere is used to preview the shader. Only one preview shape at a time can be enabled.
Preview with cube	When enabled, a model of a cube is used to preview the shader. Only one preview shape at a time can be enabled.
Preview with Cylinder	When enabled, a model of a cylinder is used to preview the shader. Only one preview shape at a time can be enabled.
Preview with cone	When enabled, a model of a cone is used to preview the shader. Only one preview shape at a time can be enabled.
Preview with teapot	When enabled, a model of a teapot is used to preview the shader. Only one preview shape at a time can be enabled.
Preview with plane	When enabled, a model of a plane is used to preview the shader. Only one preview shape at a time can be enabled.
Toolbox	Alternately shows or hides the Toolbox .
Properties	Alternatively shows or hides the Properties window.

TOOLBAR ITEM	DESCRIPTION
Advanced	<p>Contains advanced commands and options.</p> <p>Export: Enables the export of a shader in several formats.</p> <p>Export As: Exports the shader as either HLSL source code or as compiled shader bytecode. For more information about how to export shaders, see How to: Export a shader.</p> <p>Graphics Engines: Enables the selection of the renderer that is used to display the design surface.</p> <p>Render with D3D11: Uses Direct3D 11 to render the Shader Designer design surface.</p> <p>Render with D3D11WARP: Uses Direct3D 11 Windows Advanced Rasterization Platform (WARP) to render the Shader Designer design surface.</p> <p>View: Enables the selection of additional information about the Shader Designer.</p> <p>Frame Rate: When enabled, displays the current frame rate in the upper-right corner of the design surface. The frame rate is the number of frames that are drawn per second. This option is useful when you enable the Real-Time Rendering Mode option.</p>

TIP

You can choose the **Advanced** button to run the last command again.

Work with nodes and connections

Use **Select** mode to add, remove, reposition, connect, and configure nodes. Here's how to perform these basic operations:

To perform basic operations in Select mode

- Here's how:
 - To add a node to the graph, select it in the **Toolbox** and then move it to the design surface.
 - To remove a node from the graph, select it and then press **Delete**.
 - To reposition a node, select it and then move it to a new location.
 - To connect two nodes, move an output terminal of one node to an input terminal of the other node. Only terminals that have compatible types can be connected. A line between the terminals shows the connection.
 - To remove a connection, on the shortcut menu for either one of the connected terminals, choose **Break Links**.
 - To configure the properties of a node, select the node, and then, in the **Properties** window, specify new values for the properties.

Preview shaders

To help you understand how a shader will appear in your app, you can configure how your effect is previewed. To approximate your app, you can choose one of several shapes to render, configure textures and other

material parameters, enable animation of time-based effects, and examine the preview from different angles.

Shapes

The Shader Designer includes six shapes—a sphere, a cube, a cylinder, a cone, a teapot, and a plane—that you can use to preview your shader. Depending on the shader, certain shapes might give you a better preview.

To choose a preview shape, on the **Shader Designer Modes** toolbar, choose the shape that you want.

Textures and material parameters

Many shaders rely on textures and material properties to produce a unique appearance for each kind of object in your app. To see what your shader will look like in your app, you can set the textures and material properties that are used to render the preview to match the textures and parameters that you might use in your app.

To bind a different texture to a texture register, or to modify other material parameters:

1. In **Select** mode, select an empty area of the design surface. This causes the **Properties** window to display the global shader properties.
2. In the **Properties** window, specify new values for the texture and parameter properties that you want to change.

The following table shows the shader parameters that you can modify:

PARAMETER	PROPERTIES
Texture 1 - Texture 8	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Filename: The full path of the texture file that is associated with this texture register.
Material Ambient	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Value: The diffuse color of the current pixel due to indirect - or ambient - lighting.
Material Diffuse	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Value: A color that describes how the current pixel diffuses direct lighting.
Material Emissive	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Value: The color contribution of the current pixel due to self-provided lighting.
Material Specular	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Value: A color that describes how the current pixel reflects direct lighting.
Material Specular Power	Access: Public to allow the property to be set from the Model Editor; otherwise, Private . Value: The exponent that defines the intensity of specular highlights on the current pixel.

Time-based effects

Some shaders have a time-based component that animates the effect. To show what the effect looks like in action, the preview has to be updated several times per second. By default, the preview is only updated when the shader is changed; to change this behavior so that you can view time-based effects, you have to enable real-time rendering.

To enable real-time rendering, on the Shader Designer toolbar, choose **Real time Rendering**.

Examine the effect

Many shaders are affected by variables such as viewing angle or directional lighting. To examine how the effect responds as these variables change, you can rotate the preview shape freely and observe how the shader behaves.

To rotate the shape, press and hold **Alt**, and then select any point on the design surface and move it.

Export shaders

Before you can use a shader in your app, you have to export it in a format that DirectX understands.

You can export shaders as HLSL source code or as compiled shader bytecode. HLSL source code is exported to a text file that has an *.hsl* file name extension. Shader bytecode can be exported either to a raw binary file that has a *.cso* file name extension, or to a C++ header (*.h*) file that encodes the shader bytecode into an array.

For more information about how to export shaders, see [How to: Export a shader](#).

Keyboard shortcuts

COMMAND	KEYBOARD SHORTCUTS
Switch to Select mode	Ctrl+G, Ctrl+Q S
Switch to Zoom mode	Ctrl+G, Ctrl+Z Z
Switch to Pan mode	Ctrl+G, Ctrl+P K
Select all	Ctrl+A
Delete the current selection	Delete
Cancel the current selection	Escape (Esc)
Zoom in	Ctrl+Mouse wheel forward Plus Sign (+)
Zoom out	Ctrl+Mouse wheel backward Minus Sign (-)
Pan the design surface up	Mouse wheel backward PageDown

COMMAND	KEYBOARD SHORTCUTS
Pan the design surface down	Mouse wheel forward PageUp
Pan the design surface left	Shift+ Mouse wheel backward Mouse wheel left Shift+ PageDown
Pan the design surface right	Shift+ Mouse wheel forward Mouse wheel right Shift+ PageUp
Move the keyboard focus to another node	The Arrow keys
Select the node that has keyboard focus (adds the node to the selection group)	Shift+ Spacebar
Toggle selection of the node that has keyboard focus	Ctrl+ Spacebar
Toggle current selection (if no nodes are selected, select the node that has keyboard focus)	Spacebar
Move the current selection up	Shift+ Up Arrow
Move the current selection down	Shift+ Down Arrow
Move the current selection left	Shift+ Left Arrow
Move current selection right	Shift+ Right Arrow.

Related topics

TITLE	DESCRIPTION
Working with 3D assets for games and apps	Provides an overview of the Visual Studio tools that you can use to work with textures and images, 3D models, and shader effects.
Image Editor	Describes how to use the Visual Studio Image Editor to work with textures and images.
Model Editor	Describes how to use the Visual Studio Model Editor to work with 3D models.

Shader Designer nodes

2/8/2019 • 3 minutes to read • [Edit Online](#)

The articles in this section of the documentation contain information about the various Shader Designer nodes that you can use to create graphics effects.

Nodes and node types

The Shader Designer represents visual effects as a graph. These graphs are built from nodes that are specifically chosen and connected in precise ways to achieve the intended effect. Each node represents either a piece of information or a mathematical function, and the connections between them represent how the information flows through the graph to produce the result. The Shader Designer provides six different node types—filters, texture nodes, parameters, constants, utility nodes, and math nodes—and several individual nodes belong to each type. These nodes and node types are described in the other articles in this section. For more information, see the links at the end of this document.

Node structure

All nodes are made up of a combination of common elements. Every node has at least one output terminal on its right-hand side (except the final color node, which represents the output of the shader). Nodes that represent calculations or texture samplers have input terminals on their left-hand sides, but nodes that represent information have no input terminals. Output terminals are connected to input terminals to move information from one node to another.

Promotion of inputs

Because the Shader Designer must ultimately generate HLSL source code so that the effect can be used in a game or app, Shader Designer nodes are subject to the type-promotion rules that HLSL uses. Because graphics hardware operates primarily on floating-point values, type promotion between different types—for example, from `int` to `float`, or from `float` to `double`—is uncommon. Instead, because graphics hardware uses the same operation on multiple pieces of information at once, a different kind of promotion can occur in which the shorter of a number of inputs is lengthened to match the size of the longest input. How it is lengthened depends on the type of the input, and also on the operation itself:

- **If the smaller type is a scalar value, then:**

The value of the scalar is replicated into a vector that is equal in size to the larger input. For example, the scalar input 5.0 becomes the vector (5.0, 5.0, 5.0) when the largest input of the operation is a three-element vector, regardless of what the operation is.

- **If the smaller type is a vector, and the operation is multiplicative (*, /, %, and so on), then:**

The value of the vector is copied into the leading elements of a vector that is equal in size to the larger input, and the trailing elements are set to 1.0. For example, the vector input (5.0, 5.0) becomes the vector (5.0, 5.0, 1.0, 1.0) when it's multiplied by a four-element vector. This preserves the third and fourth elements of the output by using the multiplicative identity, 1.0.

- **If the smaller type is a vector, and the operation is additive (+, -, and so on), then:**

The value of the vector is copied into the leading elements of a vector that is equal in size to the larger input, and the trailing elements are set to 0.0. For example, the vector input (5.0, 5.0) becomes the vector (5.0, 5.0, 0.0, 0.0) when it's added to a four-element vector. This preserves the third and fourth elements of the output by using the additive identity, 0.0.

Related topics

TITLE	DESCRIPTION
Constant nodes	Describes nodes that you can use to represent literal values and interpolated vertex-state information in shader calculations. Because vertex-state is interpolated—and therefore, is different for each pixel—each pixel-shader instance receives a different version of the constant.
Parameter nodes	Describes nodes that you can use to represent camera position, material properties, lighting parameters, time, and other app-state information in shader calculations.
Texture nodes	Describes the nodes that you can use to sample various texture types and geometries, and to produce or transform texture coordinates in common ways.
Math nodes	Describes the nodes that you can use to perform algebraic, logic, trigonometric, and other mathematical operations that map directly to HLSL instructions.
Utility nodes	Describes the nodes that you can use to perform common lighting calculations and other common operations that do not map directly to HLSL instructions.
Filter nodes	Describes the nodes that you can use to perform texture filtering and color filtering.

Constant nodes

2/8/2019 • 3 minutes to read • [Edit Online](#)

In the Shader Designer, constant nodes represent literal values and interpolated vertex attributes in pixel-shader calculations. Because vertex attributes are interpolated—and so, are different for each pixel—each pixel-shader instance receives a different version of the constant. This gives each pixel a unique appearance.

Vertex attribute interpolation

The image of a 3D scene in a game or app is made by mathematically transforming a number of objects—which are defined by vertices, vertex attributes, and primitive definitions—into on-screen pixels. All of the information that's required to give a pixel its unique appearance is supplied through vertex attributes, which are blended together according to the pixel's proximity to the different vertices that make up its *primitive*. A primitive is a basic rendering element; that is, a simple shape such as a point, a line, or a triangle. A pixel that's very close to just one of the vertices receives constants that are nearly identical to that vertex, but a pixel that's evenly spaced between all the vertices of a primitive receives constants that are the average of those vertices. In graphics programming, the constants that the pixels receive are said to be *interpolated*. Providing constant data to pixels in this way produces very good visual quality and at the same time reduces memory footprint and bandwidth requirements.

Although each pixel-shader instance receives only one set of constant values and cannot change these values, different pixel-shader instances receive different sets of constant data. This design enables a shader program to produce a different color output for each pixel in the primitive.

Constant node reference

NODE	DETAILS	PROPERTIES
Camera Vector	<p>The vector that extends from the current pixel to the camera in world space.</p> <p>You can use this to calculate reflections in world space.</p> <p>Output</p> <p>Output : float3</p> <p>The vector from the current pixel to the camera.</p>	None
Color Constant	<p>A constant color value.</p> <p>Output</p> <p>Output : float4</p> <p>The color value.</p>	<p>Output</p> <p>The color value.</p>

NODE	DETAILS	PROPERTIES
Constant	<p>A constant scalar value.</p> <p>Output</p> <div>Output : float</div> <p>The scalar value.</p>	<p>Output</p> <p>The scalar value.</p>
2D Constant	<p>A two-component vector constant.</p> <p>Output</p> <div>Output : float2</div> <p>The vector value.</p>	<p>Output</p> <p>The vector value.</p>
3D Constant	<p>A three-component vector constant.</p> <p>Output</p> <div>Output : float3</div> <p>The vector value.</p>	<p>Output</p> <p>The vector value.</p>
4D Constant	<p>A four-component vector constant.</p> <p>Output</p> <div>Output : float4</div> <p>The color value.</p>	<p>Output</p> <p>The vector value.</p>
Normalized Position	<p>The position of the current pixel, expressed in normalized device coordinates.</p> <p>The x-coordinate and y-coordinate have values in the range of [-1, 1], the z-coordinate has a value in the range of [0, 1], and the w component contains the point depth value in view space; w is not normalized.</p> <p>Output</p> <div>Output : float4</div> <p>The position of the current pixel.</p>	None
Point Color	<p>The diffuse color of the current pixel, which is a combination of the material diffuse color and vertex color attributes.</p> <p>Output</p> <div>Output : float4</div> <p>The diffuse color of the current pixel.</p>	None

NODE	DETAILS	PROPERTIES
Point Depth	<p>The depth of the current pixel in view space.</p> <p>Output</p> <p>Output : float</p> <p>The depth of the current pixel.</p>	None
Normalized Point Depth	<p>The depth of the current pixel, expressed in normalized device coordinates.</p> <p>The result has a value in the range of [0, 1].</p> <p>Output</p> <p>Output : float</p> <p>The depth of the current pixel.</p>	None
Screen Position	<p>The position of the current pixel, expressed in screen coordinates.</p> <p>The screen coordinates are based on the current viewport. The x and y components contain the screen coordinates, the z component contains the depth normalized to a range of [0, 1], and the w component contains the depth value in view space.</p> <p>Output</p> <p>Output : float4</p> <p>The position of the current pixel.</p>	None
Surface Normal	<p>The surface normal of the current pixel in object space.</p> <p>You can use this to calculate lighting contributions and reflections in object space.</p> <p>Output</p> <p>Output : float3</p> <p>The surface normal of the current pixel.</p>	None

NODE	DETAILS	PROPERTIES
Tangent Space Camera Vector	<p>The vector that extends from the current pixel to the camera in tangent space.</p> <p>You can use this to calculate reflections in tangent space.</p> <p>Output</p> <p>Output : float3</p> <p>The vector from the current pixel to the camera.</p>	None
Tangent Space Light Direction	<p>The vector that defines the direction in which light is cast from a light source in the tangent space of the current pixel.</p> <p>You can use this to calculate lighting and specular contributions in tangent space.</p> <p>Output:</p> <p>Output : float3</p> <p>The vector from the current pixel to a light source.</p>	None
World Normal	<p>The surface normal of the current pixel in world space.</p> <p>You can use this to calculate lighting contributions and reflections in world space.</p> <p>Output</p> <p>Output : float3</p> <p>The surface normal of the current pixel.</p>	None
World Position	<p>The position of the current pixel in world space.</p> <p>Output</p> <p>Output : float4</p> <p>The position of the current pixel.</p>	None

Parameter nodes

2/8/2019 • 3 minutes to read • [Edit Online](#)

In the Shader Designer, parameter nodes represent inputs to the shader that are under the control of the app on a per-draw basis, for example, material properties, directional lights, camera position, and time. Because you can change these parameters with each draw call, you can use the same shader to give an object different appearances.

Parameter node reference

NODE	DETAILS	PROPERTIES
Camera World Position	<p>The position of the camera in world space.</p> <p>Output:</p> <p>Output : float4</p> <p>The position of the camera.</p>	None
Light Direction	<p>The vector that defines the direction in which light is cast from a light source in world space.</p> <p>You can use this to calculate lighting and specular contributions in world space.</p> <p>Output:</p> <p>Output : float3</p> <p>The vector from the current pixel to a light source.</p>	None
Material Ambient	<p>The diffuse color contribution of the current pixel that is attributed to indirect lighting.</p> <p>The diffuse color of a pixel simulates how lighting interacts with rough surfaces. You can use the Material Ambient parameter to approximate how indirect lighting contributes to the appearance of an object in the real world.</p> <p>Output:</p> <p>Output : float4</p> <p>The diffuse color of the current pixel that's due to indirect—that is, ambient—lighting.</p>	<p>Access</p> <p>Public to enable the property to be set from the Model Editor; otherwise, Private.</p> <p>Value</p> <p>The diffuse color of the current pixel that's due to indirect—that is, ambient—lighting.</p>

NODE	DETAILS	PROPERTIES
Material Diffuse	<p>A color that describes how the current pixel diffuses direct lighting.</p> <p>The diffuse color of a pixel simulates how lighting interacts with rough surfaces. You can use the Material Diffuse parameter to change how the current pixel diffuses direct lighting—that is, directional, point, and spot lights.</p> <p>Output:</p> <div>Output : float4</div> <p>A color that describes how the current pixel diffuses direct lighting.</p>	<p>Access Public to enable the property to be set from the Model Editor; otherwise, Private.</p> <p>Value A color that describes how the current pixel diffuses direct lighting.</p>
Material Emissive	<p>The color contribution of the current pixel that is attributed to lighting that it supplies to itself.</p> <p>You can use this to simulate a glowing object; that is, an object that supplies its own light. This light doesn't affect other objects.</p> <p>Output:</p> <div>Output : float4</div> <p>The color contribution of the current pixel that's due to self-provided lighting.</p>	<p>Access Public to enable the property to be set from the Model Editor; otherwise, Private.</p> <p>Value The color contribution of the current pixel that's due to self-provided lighting.</p>
Material Specular	<p>A color that describes how the current pixel reflects direct lighting.</p> <p>The specular color of a pixel simulates how lighting interacts with smooth, mirror-like surfaces. You can use the Material Specular parameter to change how the current pixel reflects direct lighting—that is, directional, point, and spot lights.</p> <p>Output:</p> <div>Output : float4</div> <p>A color that describes how the current pixel reflects direct lighting.</p>	<p>Access Public to allow the property to be set from the Model Editor; otherwise, Private.</p> <p>Value A color that describes how the current pixel reflects direct lighting.</p>

NODE	DETAILS	PROPERTIES
Material Specular Power	<p>A scalar value that describes the intensity of specular highlights.</p> <p>The larger the specular power, the more intense and far-reaching the specular highlights become.</p> <p>Output:</p> <div>Output : float</div> <p>An exponential term that describes the intensity of specular highlights on the current pixel.</p>	<p>Access</p> <p>Public to enable the property to be set from the Model Editor; otherwise, Private.</p> <p>Value</p> <p>The exponent that defines the intensity of specular highlights on the current pixel.</p>
Normalized Time	<p>The time in seconds, normalized to the range [0, 1], such that when time reaches 1, it resets to 0.</p> <p>You can use this as a parameter in shader calculations, for example, to animate texture coordinates, color values, or other attributes.</p> <p>Output:</p> <div>Output : float</div> <p>The normalized time, in seconds.</p>	None
Time	<p>The time in seconds.</p> <p>You can use this as a parameter in shader calculations, for example, to animate texture coordinates, color values, or other attributes.</p> <p>Output:</p> <div>Output : float</div> <p>The time, in seconds.</p>	None

Texture nodes

2/8/2019 • 3 minutes to read • [Edit Online](#)

In the Shader Designer, texture nodes sample various texture types and geometries, and produce or transform texture coordinates. Textures provide color and lighting detail on objects.

Texture node reference

NODE	DETAILS	PROPERTIES
Cubemap Sample	<p>Takes a color sample from a cubemap at the specified coordinates.</p> <p>You can use a cubemap to provide color detail for reflection effects, or to apply to a spherical object a texture that has less distortion than a 2D texture.</p> <p>Input:</p> <p>UVW : float3</p> <p>A vector that specifies the location on the texture cube where the sample is taken. The sample is taken where this vector intersects the cube.</p> <p>Output:</p> <p>Output : float4</p> <p>The color sample.</p>	<p>Texture</p> <p>The texture register that's associated with the sampler.</p>
Normal Map Sample	<p>Takes a normal sample from a 2D normal map at the specified coordinates</p> <p>You can use a normal map to simulate the appearance of additional geometric detail on the surface of an object. Normal maps contain packed data that represents a unit vector instead of color data</p> <p>Input:</p> <p>UV : float2</p> <p>The coordinates where the sample is taken.</p> <p>Output:</p> <p>Output : float3</p> <p>The normal sample.</p>	<p>Axis Adjustment</p> <p>The factor that's used to adjust the handedness of the normal map sample.</p> <p>Texture</p> <p>The texture register that's associated with the sampler.</p>

NODE	DETAILS	PROPERTIES
Pan UV	<p>Pans the specified texture coordinates as a function of time.</p> <p>You can use this to move a texture or normal map across the surface of an object.</p> <p>Input:</p> <p>UV : float2 The coordinates to pan.</p> <p>Time : float The length of time to pan by, in seconds.</p> <p>Output:</p> <p>Output : float2 The panned coordinates.</p>	<p>Speed X The number of texels that are panned along the x axis, per second.</p> <p>Speed Y The number of texels that are panned along the y axis, per second.</p>
Parallax UV	<p>Displaces the specified texture coordinates as a function of height and viewing angle.</p> <p>The effect this creates is known as <i>parallax mapping</i>, or virtual displacement mapping. You can use it to create an illusion of depth on a flat surface.</p> <p>Input:</p> <p>UV : float2 The coordinates to displace.</p> <p>Height : float The heightmap value that's associated with the UV coordinates.</p> <p>Output:</p> <p>Output : float2 The displaced coordinates.</p>	<p>Depth Plane The reference depth for the parallax effect. By default, the value is 0.5. Smaller values lift the texture; larger values sink it into the surface.</p> <p>Depth Scale The scale of the parallax effect. This makes the apparent depth more or less pronounced. Typical values range from 0.02 to 0.1.</p>

NODE	DETAILS	PROPERTIES
Rotate UV	<p>Rotates the specified texture coordinates around a central point as a function of time.</p> <p>You can use this to spin a texture or normal map on the surface of an object.</p> <p>Input:</p> <p>UV : float2 The coordinates to rotate.</p> <p>Time : float The length of time to pan by, in seconds.</p> <p>Output:</p> <p>Output : float2 The rotated coordinates.</p>	<p>Center X The x coordinate that defines the center of rotation.</p> <p>Center Y The y coordinate that defines the center of rotation.</p> <p>Speed The angle, in radians, by which the texture rotates per second.</p>
Texture Coordinate	<p>The texture coordinates of the current pixel.</p> <p>The texture coordinates are determined by interpolating among the texture coordinate attributes of nearby vertices. You can think of this as the position of the current pixel in texture space.</p> <p>Output:</p> <p>Output : float2 The texture coordinates.</p>	None
Texture Dimensions	<p>Outputs the width and height of a 2D texture map.</p> <p>You can use the texture dimensions to consider the width and height of the texture in a shader.</p> <p>Output:</p> <p>Output : float2 The width and height of the texture, expressed as a vector. The width is stored in the first element of the vector. The height is stored in the second element.</p>	<p>Texture The texture register that's associated with the texture dimensions.</p>

NODE	DETAILS	PROPERTIES
Texel Delta	<p>Outputs the delta (distance) between the texels of a 2D texture map.</p> <p>You can use the texel delta to sample neighboring texel values in a shader.</p> <p>Output:</p> <p>Output : float2</p> <p>The delta (distance) from a texel to the next texel (moving diagonally in the positive direction), expressed as a vector in normalized texture space. You can derive the positions of all neighboring texels by selectively ignoring or negating the U or V coordinates of the delta.</p>	<p>Texture</p> <p>The texture register that's associated with the texel delta.</p>
Texture Sample	<p>Takes a color sample from a 2D texture map at the specified coordinates.</p> <p>You can use a texture map to provide color detail on the surface of an object.</p> <p>Input:</p> <p>UV : float2</p> <p>The coordinates where the sample is taken.</p> <p>Output:</p> <p>Output : float4</p> <p>The color sample.</p>	<p>Texture</p> <p>The texture register that's associated with the sampler.</p>

Math nodes

2/8/2019 • 9 minutes to read • [Edit Online](#)

In the Shader Designer, math nodes perform algebraic, logic, trigonometric, and other mathematical operations.

NOTE

When you work with math nodes in the Shader Designer, type promotion is especially evident. To learn how type promotion affects input parameters, see the "Promotion of inputs" section in [Shader Designer nodes](#).

Math node reference

NODE	DETAILS	PROPERTIES
Abs	<p>Computes the absolute value of the specified input per component.</p> <p>For each component of input <code>x</code>, negative values are made positive so that every component of the result has a positive value.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code></p> <p>The values for which to determine the absolute value.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code></p> <p>The absolute value, per component.</p>	None
Add	<p>Computes the component-wise sum of the specified inputs per component.</p> <p>For each component of the result, the corresponding components of input <code>x</code> and input <code>y</code> are added together.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code></p> <p>One of the values to add together.</p> <p><code>y</code> : same as input <code>x</code></p> <p>One of the values to add together.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code></p> <p>The sum, per component.</p>	None

NODE	DETAILS	PROPERTIES
Ceil	<p>Computes the ceiling of the specified input per component.</p> <p>The ceiling of a value is the smallest integer that's greater than or equal to that value.</p> <p>Input:</p> <p><code>x</code> : <code>float</code> , <code>float2</code> , <code>float3</code> , or <code>float4</code></p> <p>The values for which to compute the ceiling.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code></p> <p>The ceiling, per component.</p>	None
Clamp	<p>Clamps each component of the specified input to a predefined range.</p> <p>For each component of the result, values that are below the defined range are made equal to the minimum value in the range, values that are above the defined range are made equal to the maximum value in the range, and values that are in the range are not changed.</p> <p>Input:</p> <p><code>x</code> : <code>float</code> , <code>float2</code> , <code>float3</code> , or <code>float4</code></p> <p>The values to clamp.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code></p> <p>The clamped value, per component.</p>	<p>Max</p> <p>The largest possible value in the range.</p> <p>Min</p> <p>The smallest possible value in the range.</p>

NODE	DETAILS	PROPERTIES
Cos	<p>Computes the cosine of the specified input, in radians, per component.</p> <p>For each component of the result, the cosine of the corresponding component, which is provided in radians, is calculated. The result has components that have values in the range of [-1, 1].</p> <p>Input:</p> <p>X : float, float2, float3, or float4</p> <p>The values to compute the cosine of, in radians.</p> <p>Output:</p> <p>Output : same as input X</p> <p>The cosine, per component.</p>	None
Cross	<p>Computes the cross product of the specified three-component vectors.</p> <p>You can use the cross product to compute the normal of a surface that's defined by two vectors.</p> <p>Input:</p> <p>X : float3</p> <p>The vector on the left-hand-side of the cross product.</p> <p>Y : float3</p> <p>The vector on the right-hand-side of the cross product.</p> <p>Output:</p> <p>Output : float3</p> <p>The cross product.</p>	None

NODE	DETAILS	PROPERTIES
Distance	<p>Computes the distance between the specified points.</p> <p>The result is a positive scalar value.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> One of the points to determine the distance between.</p> <p><code>y</code> : same as input <code>x</code> One of the points to determine the distance between.</p> <p>Output:</p> <p><code>output</code> : same as input <code>x</code> The distance.</p>	None
Divide	<p>Computes the component-wise quotient of the specified inputs.</p> <p>For each component of the result, the corresponding component of input <code>x</code> is divided by the corresponding component of input <code>y</code>.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> The dividend values.</p> <p><code>y</code> : same as input <code>x</code> The divisor values.</p> <p>Output:</p> <p><code>output</code> : same as input <code>x</code> The quotient, per component.</p>	None

NODE	DETAILS	PROPERTIES
Dot	<p>Computes the dot product of the specified vectors.</p> <p>The result is a scalar value. You can use the dot product to determine the angle between two vectors.</p> <p>Input:</p> <p><code>x</code> : <code>float</code> , <code>float2</code> , <code>float3</code> , or <code>float4</code> One of the terms.</p> <p><code>y</code> : same as input <code>x</code> One of the terms.</p> <p>Output:</p> <p><code>Output</code> : <code>float</code> The dot product.</p>	None
Floor	<p>Computes the floor of the specified input per component.</p> <p>For each component of the result, its value is the largest whole integer value that's less than or equal to the corresponding component of the input. Every component of the result is a whole integer.</p> <p>Input:</p> <p><code>x</code> : <code>float</code> , <code>float2</code> , <code>float3</code> , or <code>float4</code> The values for which to compute the floor.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The floor, per component.</p>	None

NODE	DETAILS	PROPERTIES
Fmod	<p>Computes the component-wise modulus (remainder) of the specified inputs.</p> <p>For each component of the result, some integral (whole-number) multiple, m, of the corresponding component of input y is subtracted from the corresponding component of input x, leaving a remainder. The multiple, m, is chosen such that the remainder is less than the corresponding component of input y and has the same sign as the corresponding component of input x. For example, $\text{fmod}(-3.14, 1.5)$ is -0.14.</p> <p>Input:</p> <p>x : float, float2, float3, or float4 The dividend values.</p> <p>y : same as input x The divisor values.</p> <p>Output:</p> <p>Output : same as input x The modulus, per component.</p>	None
Frac	<p>Removes the integral (whole-number) part of the specified input per component.</p> <p>For each component of the result, the integral part of the corresponding component of the input is removed, but the fractional part and sign are retained. This fractional value falls in the range $[0, 1)$. For example, the value -3.14 becomes the value -0.14.</p> <p>Input:</p> <p>x : float, float2, float3, or float4 The values for which to compute the fractional part.</p> <p>Output:</p> <p>Output : same as input x The fractional part, per component.</p>	None

NODE	DETAILS	PROPERTIES
Lerp	<p>Linear Interpolation. Computes the component-wise weighted average of the specified inputs.</p> <p>For each component of the result, the weighted average of the corresponding components of the inputs <code>x</code> and <code>y</code>. The weight is provided by <code>Percent</code>, a scalar, and is uniformly applied to all components. You can use this to interpolate between points, colors, attributes, and other values.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, Or <code>float4</code> The originating value. When <code>Percent</code> is zero, the result is equal to this input.</p> <p><code>y</code> : same as input <code>x</code> The terminal value. When <code>Percent</code> is one, the result is equal to this input.</p> <p><code>Percent</code> : <code>float</code> A scalar weight that's expressed as a percentage of the distance from input <code>x</code> towards input <code>y</code>.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> A value that's collinear with the specified inputs.</p>	None

NODE	DETAILS	PROPERTIES
Multiply Add	<p>Computes the component-wise multiply-add of the specified inputs.</p> <p>For each component of the result, the product of the corresponding components of the inputs <code>M</code> and <code>A</code> is added to the corresponding component of input <code>B</code>. This operation sequence is found in common formulas—for example, in the point-slope formula of a line, and in the formula to scale and then bias an input.</p> <p>Input:</p> <p><code>M</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> One of the values to multiply together.</p> <p><code>A</code> : same as input <code>M</code> One of the values to multiply together.</p> <p><code>B</code> : same as input <code>M</code> The values to add to the product of the other two inputs.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>M</code> The result of the multiply-add, per component.</p>	None
Max	<p>Computes the component-wise maximum of the specified inputs.</p> <p>For each component of the result, the greater of the corresponding components of the inputs is taken.</p> <p>Input:</p> <p><code>X</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> One of the values for which to compute the maximum.</p> <p><code>Y</code> : same as input <code>X</code> One of the values for which to compute the maximum.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>X</code> The maximum value, per component.</p>	None

NODE	DETAILS	PROPERTIES
Min	<p>Computes the component-wise minimum of the specified inputs.</p> <p>For each component of the result, the lesser of the corresponding components of the inputs is taken.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> One of the values for which to compute the minimum.</p> <p><code>y</code> : same as input <code>x</code> One of the values for which to compute the minimum.</p> <p>Output:</p> <p><code>output</code> : same as input <code>x</code> The minimum value, per component.</p>	None
Multiply	<p>Computes the component-wise product of the specified inputs.</p> <p>For each component of the result, the corresponding components of the inputs <code>x</code> and <code>y</code> are multiplied together.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> One of the values to multiply together.</p> <p><code>y</code> : same as input <code>x</code> One of the values to multiply together.</p> <p>Output:</p> <p><code>output</code> : same as input <code>x</code> The product, per component.</p>	None

NODE	DETAILS	PROPERTIES
Normalize	<p>Normalizes the specified vector.</p> <p>A normalized vector retains the direction of the original vector, but not its magnitude. You can use normalized vectors to simplify calculations where the magnitude of a vector is not important.</p> <p>Input:</p> <p><code>x</code> : <code>float2</code>, <code>float3</code>, or <code>float4</code> The vector to normalize.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The normalized vector.</p>	None
One Minus	<p>Computes the difference between 1 and the specified input per component.</p> <p>For each component of the result, the corresponding component of the input is subtracted from 1.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> The values to be subtracted from 1.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The difference between 1 and the specified input, per component.</p>	None

NODE	DETAILS	PROPERTIES
Power	<p>Computes the component-wise exponentiation (power) of the specified inputs.</p> <p>For each component of the result, the corresponding component of input <code>x</code> is raised to the power of the corresponding component of the input <code>y</code>.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, Or <code>float4</code> The base values</p> <p><code>y</code> : same as input <code>x</code> The exponent values.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The exponentiation, per component.</p>	None
Saturate	<p>Clamps each component of the specified input to the range [0, 1].</p> <p>You can use this range to represent percentages and other relative measurements in calculations. For each component of the result, the corresponding component values of the input that are less than 0 are made equal to 0, values that are larger than 1 are made equal to 1, and values that are in the range are not changed.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, Or <code>float4</code> The values to saturate.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The saturated value, per component.</p>	None

NODE	DETAILS	PROPERTIES
Sin	<p>Computes the sine of the specified input, in radians, per component.</p> <p>For each component of the result, the sine of the corresponding component, which is provided in radians, is calculated. The result has components that have values in the range [-1, 1].</p> <p>Input:</p> <p>X : float, float2, float3, or float4</p> <p>The values to compute the sine of, in radians.</p> <p>Output:</p> <p>Output : same as input x</p> <p>The sine, per component.</p>	None
Sqrt	<p>Computes the square root of the specified input, per component.</p> <p>For each component of the result, the square root of the corresponding component is calculated.</p> <p>Input:</p> <p>X : float, float2, float3, or float4</p> <p>The values for which to compute the square root.</p> <p>Output:</p> <p>Output : same as input x</p> <p>The square root, per component.</p>	None

NODE	DETAILS	PROPERTIES
Subtract	<p>Computes the component-wise difference of the specified inputs.</p> <p>For each component of the result, the corresponding component of input <code>y</code> is subtracted from the corresponding component of input <code>x</code>. You can use this to compute the vector that extends from the first input to the second.</p> <p>Input:</p> <p><code>x</code> : <code>float</code>, <code>float2</code>, <code>float3</code>, or <code>float4</code> The values to be subtracted from.</p> <p><code>y</code> : same as input <code>x</code> The values to subtract from input <code>x</code>.</p> <p>Output:</p> <p><code>Output</code> : same as input <code>x</code> The difference, per component.</p>	None
Transform 3D Vector	<p>Transforms the specified 3D vector into a different space.</p> <p>You can use this to bring points or vectors into a common space so that you can use them to perform meaningful calculations.</p> <p>Input:</p> <p><code>Vector</code> : <code>float3</code> The vector to transform.</p> <p>Output:</p> <p><code>Output</code> : <code>float3</code> The transformed vector.</p>	<p>From System The native space of the vector.</p> <p>To System The space to transform the vector into.</p>

Utility nodes

2/8/2019 • 3 minutes to read • [Edit Online](#)

In the Shader Designer, utility nodes represent common, useful shader calculations that don't fit neatly into the other categories. Some utility nodes perform simple operations such as appending vectors together or choosing results conditionally, and others perform complex operations such as computing lighting contributions according to popular lighting models.

Utility node reference

NODE	DETAILS	PROPERTIES
Append Vector	<p>Creates a vector by appending the specified inputs together.</p> <p>Input:</p> <p>Vector : float , float2 , or float3 The values to append to.</p> <p>Value to Append : float The value to append.</p> <p>Output:</p> <p>Output : float2 , float3 , or float4 depending on the type of input Vector The new vector.</p>	None

NODE	DETAILS	PROPERTIES
Fresnel	<p>Computes the Fresnel fall-off based on the specified surface normal.</p> <p>The Fresnel fall-off value expresses how closely the surface normal of the current pixel coincides with the view vector. When the vectors are aligned, the result of the function is 0; the result increases as the vectors become less similar, and reaches its maximum when the vectors are orthogonal. You can use this to make an effect more or less apparent based on the relationship between the orientation of the current pixel and the camera.</p> <p>Input:</p> <p>Surface Normal : float3</p> <p>The surface normal of the current pixel, defined in the current pixel's tangent space. You can use this to perturb the apparent surface normal, as in normal mapping.</p> <p>Output:</p> <p>Output : float</p> <p>The reflectivity of the current pixel.</p>	<p>Exponent</p> <p>The exponent that's used to calculate the Fresnel fall-off.</p>

NODE	DETAILS	PROPERTIES
If	<p>Conditionally chooses one of three potential results per component. The condition is defined by the relationship between two other specified inputs.</p> <p>For each component of the result, the corresponding component of one of the three potential results is chosen, based on the relationship between the corresponding components of the first two inputs.</p> <p>Input:</p> <p>x : float, float2, float3, or float4 The left-hand side value to compare.</p> <p>y : same type as input x The right-hand side value to compare.</p> <p>$x > y$: same type as input x The values that are chosen when x is greater than y.</p> <p>$x = y$: same type as input x The values that are chosen when x is equal to y.</p> <p>$x < y$: same type as input x The values that are chosen when x is less than y.</p> <p>Output:</p> <p>Output : float3 The chosen result, per component.</p>	None

NODE	DETAILS	PROPERTIES
Lambert	<p>Computes the color of the current pixel according to the Lambert lighting model, by using the specified surface normal.</p> <p>This color is the sum of ambient color and diffuse lighting contributions under direct lighting. Ambient color approximates the total contribution of indirect lighting, but looks flat and dull without the help of additional lighting. Diffuse lighting helps add shape and depth to an object.</p> <p>Input:</p> <p>Surface Normal : float3</p> <p>The surface normal of the current pixel, defined in the current pixel's tangent space. You can use this to perturb the apparent surface normal, as in normal mapping.</p> <p>Diffuse Color : float3</p> <p>The diffuse color of the current pixel, typically the Point Color. If no input is provided, the default value is white.</p> <p>Output:</p> <p>Output : float3</p> <p>The diffuse color of the current pixel.</p>	None
Mask Vector	<p>Masks components of the specified vector.</p> <p>You can use this to remove specific color channels from a color value, or to prevent specific components from having an effect on subsequent calculations.</p> <p>Input:</p> <p>Vector : float4</p> <p>The vector to mask.</p> <p>Output:</p> <p>Output : float4</p> <p>The masked vector.</p>	<p>Red / X False to mask out the red (x) component; otherwise, True.</p> <p>Green / Y False to mask out the green (y) component; otherwise, True.</p> <p>Blue / Z False to mask out the blue (z) component; otherwise, True.</p> <p>Alpha / W False to mask out the alpha (w) component; otherwise, True.</p>

NODE	DETAILS	PROPERTIES
Reflection Vector	<p>Computes the reflection vector for the current pixel in tangent space, based on the camera position.</p> <p>You can use this to calculate reflections, cubemap coordinates, and specular lighting contributions</p> <p>Input:</p> <p>Tangent Space Surface Normal : float3</p> <p>The surface normal of the current pixel, defined in the current pixel's tangent space. You can use this to perturb the apparent surface normal, as in normal mapping.</p> <p>Output:</p> <p>Output : float3</p> <p>The reflection vector.</p>	None
Specular	<p>Computes the specular lighting contribution according to the Phong lighting model, by using the specified surface normal.</p> <p>Specular lighting gives a shiny, reflective appearance to an object, for example, water, plastic, or metals.</p> <p>Input:</p> <p>Surface Normal : float3</p> <p>The surface normal of the current pixel, defined in the current pixel's tangent space. You can use this to perturb the apparent surface normal, as in normal mapping.</p> <p>Output:</p> <p>Output : float3</p> <p>The color contribution of specular highlights.</p>	None

Filter nodes

2/8/2019 • 2 minutes to read • [Edit Online](#)

In the Shader Designer, filter nodes transform an input—for example, a color or texture sample—into a figurative color value. These figurative color values are commonly used in non-photorealistic rendering or as components in other visual effects.

Filter node reference

NODE	DETAILS	PROPERTIES
Blur	<p>Blurs pixels in a texture by using a Gaussian function.</p> <p>You can use this to reduce color detail or noise in a texture.</p> <p>Input:</p> <p>UV : float2 The coordinates of the texel to test.</p> <p>Output:</p> <p>Output : float4 The blurred color value.</p>	<p>Texture</p> <p>The texture register that's associated with the sampler that's used during blurring.</p>
Desaturate	<p>Reduces the amount of color in the specified color.</p> <p>As color is removed, the color value approaches its gray-scale equivalent.</p> <p>Input:</p> <p>RGB : float3 The color to desaturate.</p> <p>Percent : float The percent of color to remove, expressed as a normalized value in the range [0, 1].</p> <p>Output:</p> <p>Output : float3 The desaturated color.</p>	<p>Luminance</p> <p>The weights that are given to the red, green, and blue color components.</p>

NODE	DETAILS	PROPERTIES
Edge Detection	<p>Detects edges in a texture by using a Canny edge detector. Edge pixels are output as white; non-edge pixels are output as black.</p> <p>You can use this to identify edges in a texture so that you can use additional effects to treat edge pixels.</p> <p>Input:</p> <div>UV : float2</div> <p>The coordinates of the texel to test.</p> <p>Output:</p> <div>Output : float4</div> <p>White if the texel is on an edge; otherwise, black.</p>	<p>Texture</p> <p>The texture register that's associated with the sampler that's used during edge detection.</p>
Sharpen	<p>Sharpens a texture.</p> <p>You can use this to highlight fine details in a texture.</p> <p>Input:</p> <div>UV : float2</div> <p>The coordinates of the texel to test.</p> <p>Output:</p> <div>Output : float4</div> <p>The blurred color value.</p>	<p>Texture</p> <p>The texture register that's associated with the sampler that's used during sharpening.</p>

Shader Designer examples

2/8/2019 • 2 minutes to read • [Edit Online](#)

The articles in this section of the documentation contain examples that demonstrate how you can use the Shader Designer to create various graphics effects.

Related topics

How to: Create a basic color shader	Demonstrates a shader that applies a constant color to an object.
How to: Create a basic lambert shader	Demonstrates a shader that applies the classic Lambert lighting model to an object.
How to: Create a basic phong shader	Demonstrates a shader that applies the classic Phong lighting model to an object.
How to: Create a basic texture shader	Demonstrates a shader that applies a texture to an object.
How to: Create a grayscale texture shader	Demonstrates a shader that converts a texture to grayscale during rendering and applies it to an object.
How to: Create a geometry-based gradient shader	Demonstrates a shader that modulates a color based on the geometry of an object and applies it to the object.
Walkthrough: Creating a realistic 3D billiard ball	Demonstrates how to combine shader techniques and texture resources to create a realistic billiard ball shader.
How to: Export a shader	Explains how to export a DGSL shader in a format that your app can use.

How to: Create a basic color shader

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language (DGSL) to create a flat color shader. This shader sets the final color to a constant RGB color value.

Create a flat color shader

You can implement a flat color shader by writing the color value of an RGB color constant to the final output color.

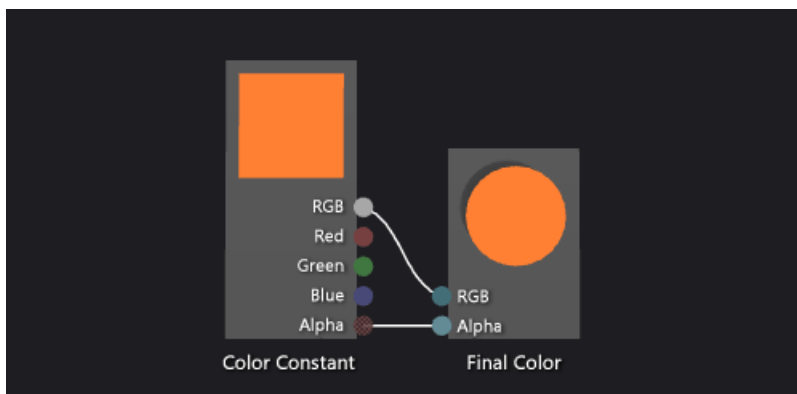
Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a DGSL shader to work with. For information about how to add a DGSL shader to your project, see the Getting Started section in [Shader Designer](#).
2. Delete the **Point Color** node. Use the **Select** tool to select the **Point Color** node, and then on the menu bar, choose **Edit > Delete**.
3. Add a **Color Constant** node to the graph. In the **Toolbox**, under **Constants**, select **Color Constant** and move it to the design surface.
4. Specify a color value for the **Color Constant** node. Use the **Select** tool to select the **Color Constant** node, and then, in the **Properties** window, in the **Output** property, specify a color value. For orange, specify a value of (1.0, 0.5, 0.2, 1.0).
5. Connect the color constant to the final color. To create the connections, move the **RGB** terminal of the **Color Constant** node to the **RGB** terminal of the **Final Color** node, and then move the **Alpha** terminal of the **Color Constant** node to the **Alpha** terminal of the **Final Color** node. These connections set the final color to the color constant defined in the previous step.

The following illustration shows the completed shader graph and a preview of the shader applied to a cube.

NOTE

In the illustration, an orange color was specified to better demonstrate the effect of the shader.



Certain shapes might provide better previews for some shaders. For more information about how to preview shaders in the Shader Designer, see [Shader Designer](#).

See also

- [How to: Apply a shader to a 3D model](#)
- [How to: Export a shader](#)
- [Shader Designer](#)
- [Shader Designer nodes](#)

How to: Create a basic Lambert shader

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language (DGSL) to create a lighting shader that implements the classic Lambert lighting model.

The Lambert lighting model

The Lambert lighting model incorporates ambient and directional lighting to shade objects in a 3D scene. The ambient components provide a base level of illumination in the 3D scene. The directional components provide additional illumination from directional (far-away) light sources. Ambient illumination affects all surfaces in the scene equally, regardless of their orientation. For a given surface, it's a product of the ambient color of the surface and the color and intensity of ambient lighting in the scene. Directional lighting affects every surface in the scene differently, based on the orientation of the surface with respect to the direction of the light source. It's a product of the diffuse color and orientation of the surface, and the color, intensity, and direction of the light sources. Surfaces that face directly toward the light source receive the maximum contribution and surfaces that face directly away receive no contribution. Under the Lambert lighting model, the ambient component and one or more directional components are combined to determine the total diffuse color contribution for each point on the object.

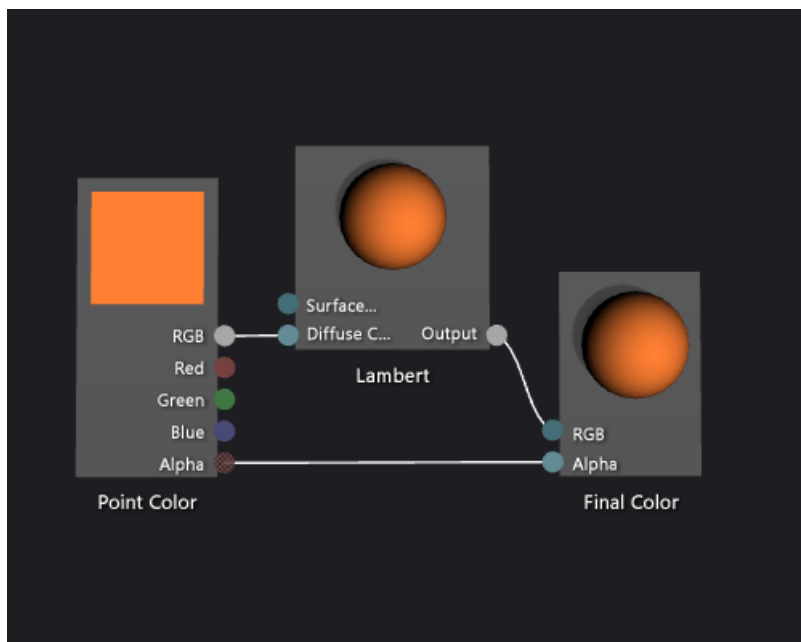
Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a DGSL shader with which to work. For information about how to add a DGSL shader to your project, see the Getting Started section in [Shader Designer](#).
2. Disconnect the **Point Color** node from the **Final Color** node. Choose the **RGB** terminal of the **Point Color** node, and then choose **Break Links**. Leave the **Alpha** terminal connected.
3. Add a **Lambert** node to the graph. In the **Toolbox**, under **Utility**, select **Lambert** and move it to the design surface. The **Lambert** node computes the total diffuse color contribution of the pixel, based on ambient and diffuse lighting parameters.
4. Connect the **Point Color** node to the **Lambert** node. In **Select** mode, move the **RGB** terminal of the **Point Color** node to the **Diffuse Color** terminal of the **Lambert** node. This connection provides the **Lambert** node with the interpolated diffuse color of the pixel.
5. Connect the computed color value to the final color. Move the **Output** terminal of the **Lambert** node to the **RGB** terminal of the **Final Color** node.

The following illustration shows the completed shader graph and a preview of the shader applied to a teapot model.

NOTE

To better demonstrate the effect of the shader in this illustration, an orange color has been specified by using the **MaterialDiffuse** parameter of the shader. A game or app can use this parameter to supply a unique color value for each object. For information about material parameters, see the Previewing Shaders section in [Shader Designer](#).



Certain shapes might provide better previews for some shaders. For more information about how to preview shaders in the Shader Designer, see the [Previewing Shaders](#) section in [Shader Designer](#).

The following illustration shows the shader that's described in this document applied to a 3D model.



For more information about how to apply a shader to a 3D model, see [How to: Apply a Shader to a 3D Model](#).

See also

- [How to: Apply a Shader to a 3D Model](#)
- [How to: Export a Shader](#)
- [How to: Create a Basic Phong Shader](#)
- [Shader Designer](#)
- [Shader Designer Nodes](#)

How to: Create a basic Phong shader

2/8/2019 • 3 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language (DGSL) to create a lighting shader that implements the classic Phong lighting model.

The Phong lighting model

The Phong lighting model extends the Lambert lighting model to include specular highlighting, which simulates the reflective properties of a surface. The specular component provides additional illumination from the same directional light sources that are used in the Lambert lighting model, but its contribution to the final color is processed differently. Specular highlighting affects every surface in the scene differently, based on the relationship between the view direction, the direction of the light sources, and the orientation of the surface. It's a product of the specular color, specular power, and orientation of the surface, and the color, intensity, and direction of the light sources. Surfaces that reflect the light source directly at the viewer receive the maximum specular contribution and surfaces that reflect the light source away from the viewer receive no contribution. Under the Phong lighting model, one or more specular components are combined to determine the color and intensity of specular highlighting for each point on the object, and then are added to the result of the Lambert lighting model to produce the final color of the pixel.

For more information about the Lambert lighting model, see [How to: Create a basic Lambert shader](#).

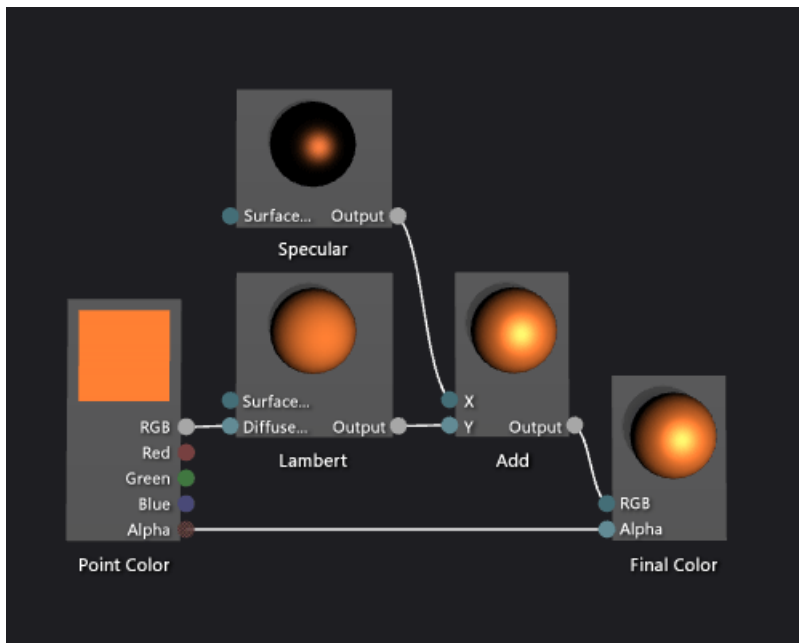
Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a Lambert shader, as described in [How to: Create a basic Lambert shader](#).
2. Disconnect the **Lambert** node from the **Final Color** node. Choose the **RGB** terminal of the **Lambert** node, and then choose **Break Links**. This makes room for the node that's added in the next step.
3. Add an **Add** node to the graph. In the **Toolbox**, under **Math**, select **Add** and move it to the design surface.
4. Add a **Specular** node to the graph. In the **Toolbox**, under **Utility**, select **Specular** and move it to the design surface.
5. Add the specular contribution. Move the **Output** terminal of the **Specular** node to the **X** terminal of the **Add** node, and then move the **Output** terminal of the **Lambert** node to the **Y** terminal of the **Add** node. These connections combine the total diffuse and specular color contributions for the pixel.
6. Connect the computed color value to the final color. Move the **Output** terminal of the **Add** node to the **RGB** terminal of the **Final Color** node.

The following illustration shows the completed shader graph and a preview of the shader applied to a teapot model.

NOTE

To better demonstrate the effect of the shader in this illustration, an orange color has been specified by using the **MaterialDiffuse** parameter of the shader, and a metallic-looking finish has been specified by using the **MaterialSpecular** and **MaterialSpecularPower** parameters. For information about material parameters, see the [Previewing Shaders](#) section in [Shader Designer](#).



Certain shapes might provide better previews for some shaders. For more information about how to preview shaders in the Shader Designer, see the [Previewing Shaders](#) section in [Shader Designer](#)

The following illustration shows the shader that's described in this document applied to a 3D model. The **MaterialSpecular** property is set to (1.00, 0.50, 0.20, 0.00), and its **MaterialSpecularPower** property is set to 16.

NOTE

The **MaterialSpecular** property determines the apparent finish of the surface material. A high-gloss surface such as glass or plastic tends to have a specular color that is a bright shade of white. A metallic surface tends to have a specular color that is close to its diffuse color. A satin-finish surface tends to have a specular color that is a dark shade of gray.

The **MaterialSpecularPower** property determines how intense the specular highlights are. High specular powers simulate duller, more-localized highlights. Very low specular powers simulate intense, sweeping highlights that can oversaturate and conceal the color of the whole surface.



For more information about how to apply a shader to a 3D model, see [How to: Apply a shader to a 3D model](#).

See also

- [How to: Apply a shader to a 3D model](#)
- [How to: Export a shader](#)
- [How to: Create a basic Lambert shader](#)
- [Shader Designer](#)
- [Shader Designer nodes](#)

How to: Create a basic texture shader

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language (DGSL) to create a single-texture shader. This shader sets the final color directly to the RGB and alpha values that are sampled from the texture.

Create a basic texture shader

You can implement a basic, single-texture shader by writing the color and alpha values of a texture sample directly to the final output color.

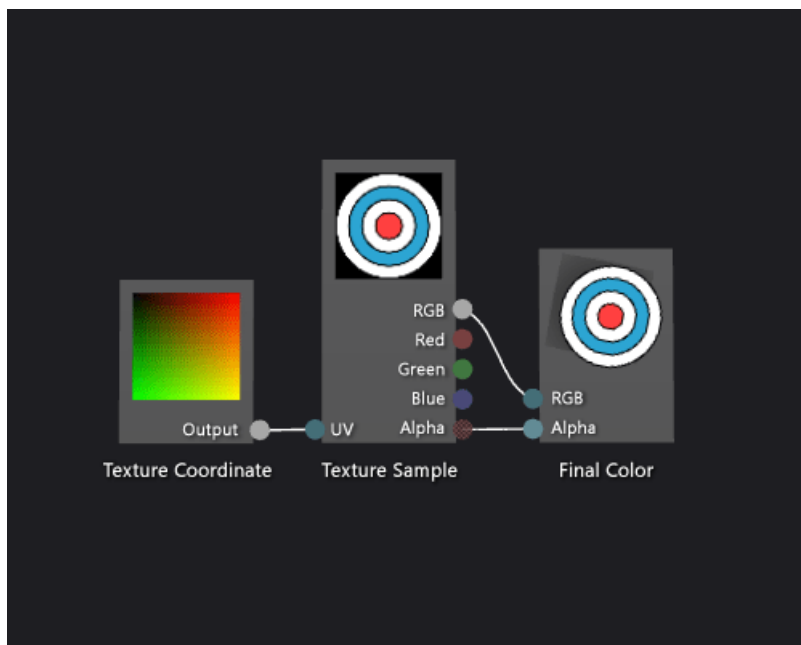
Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a DGSL shader to work with. For information about how to add a DGSL shader to your project, see the Getting Started section in [Shader Designer](#).
2. Delete the **Point Color** node. In **Select** mode, select the **Point Color** node, and then on the menu bar, choose **Edit > Delete**. This makes room for the node that's added in the next step.
3. Add a **Texture Sample** node to the graph. In the **Toolbox**, under **Texture**, select **Texture Sample** and move it to the design surface.
4. Add a **Texture Coordinate** node to the graph. In the **Toolbox**, under **Texture**, select **Texture Coordinate** and move it to the design surface.
5. Choose a texture to apply. In **Select** mode, select the **Texture Sample** node, and then in the **Properties** window, specify the texture that you want to use by using the **Filename** property.
6. Make the texture publicly accessible. Select the **Texture Sample** node, and then in the **Properties** window, set the **Access** property to **Public**. Now you can set the texture from another tool, such as the **Model Editor**.
7. Connect the texture coordinates to the texture sample. In **Select** mode, move the **Output** terminal of the **Texture Coordinate** node to the **UV** terminal of the **Texture Sample** node. This connection samples the texture at the specified coordinates.
8. Connect the texture sample to the final color. Move the **RGB** terminal of the **Texture Sample** node to the **RGB** terminal of the **Final Color** node, and then move the **Alpha** terminal of the **Texture Sample** node to the **Alpha** terminal of the **Final Color** node.

The following illustration shows the completed shader graph and a preview of the shader applied to a cube.

NOTE

In this illustration, a plane is used as the preview shape, and a texture has been specified to better demonstrate the effect of the shader.



Certain shapes might provide better previews for some shaders. For more information about how to preview shaders in the Shader Designer, see [Shader Designer](#)

See also

- [How to: Apply a shader to a 3D model](#)
- [Image Editor](#)
- [Shader Designer](#)
- [Shader Designer nodes](#)

How to: Create a grayscale texture shader

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language (DGSL) to create a grayscale texture shader. This shader modifies the RGB color value of the texture sample, and then uses it together with the unmodified alpha value to set the final color.

Create a grayscale texture shader

You can implement a grayscale texture shader by modifying the color value of a texture sample before you write it to the final output color.

Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a basic texture shader, as described in [How to: Create a basic texture shader](#).
2. Disconnect the **RGB** terminal of the **Texture Sample** node from the **RGB** terminal of the **Final Color** node. In **Select** mode, choose the **RGB** terminal of the **Texture Sample** node, and then choose **Break Links**. This makes room for the node that's added in the next step.
3. Add a **Desaturate** node to the graph. In the **Toolbox**, under **Filters**, select **Desaturate** and move it to the design surface.
4. Calculate the grayscale value by using the **Desaturate** node. In **Select** mode, move the **RGB** terminal of the **Texture Sample** node to the **RGB** terminal of the **Desaturate** node.

NOTE

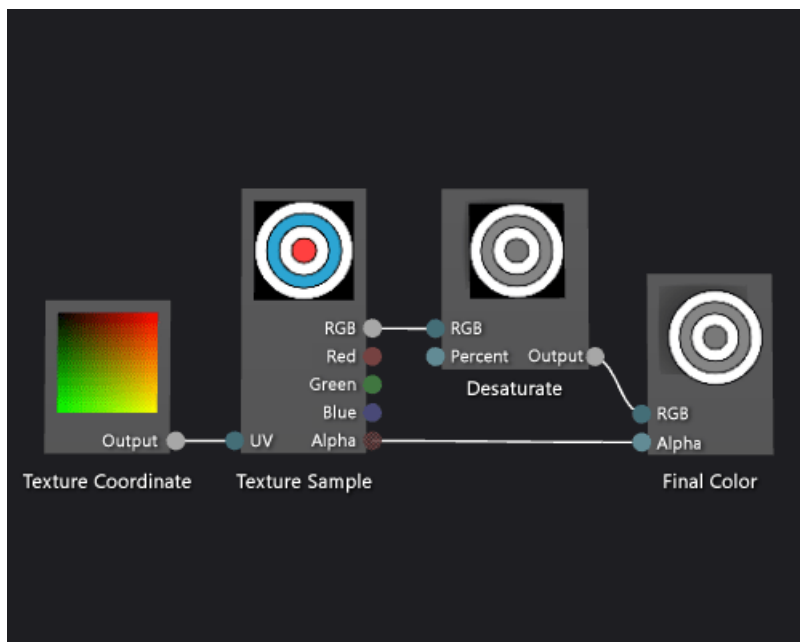
By default, the **Desaturate** node fully desaturates the input color, and uses the standard luminance weights for greyscale conversion. You can change how the **Desaturate** node behaves by changing the value of the **Luminance** property, or by only partially desaturating the input color. To partially desaturate the input color, provide a scalar value in the range [0,1) to the **Percent** terminal of the **Desaturate** node.

5. Connect the grayscale color value to the final color. Move the **Output** terminal of the **Desaturate** node to the **RGB** terminal of the **Final Color** node.

The following illustration shows the completed shader graph and a preview of the shader applied to a cube.

NOTE

In this illustration, a plane is used as the preview shape, and a texture has been specified to better demonstrate the effect of the shader.



Certain shapes might provide better previews for some shaders. For more information about previewing shaders in the Shader Designer, see [Shader Designer](#)

See also

- [How to: Apply a shader to a 3D model](#)
- [How to: Export a shader](#)
- [Image Editor](#)
- [Shader Designer](#)
- [Shader Designer nodes](#)

How to: Create a geometry-based gradient shader

2/8/2019 • 3 minutes to read • [Edit Online](#)

This article demonstrates how to use the Shader Designer and the Directed Graph Shader Language to create a geometry-based gradient shader. This shader scales a constant RGB color value by the height of each point of an object in world space.

Create a geometry-based gradient shader

You can implement a geometry-based shader by incorporating the position of the pixel into your shader. In shading languages, a pixel contains more information than just its color and location on a 2D screen. A pixel—known as a *fragment* in some systems—is a collection of values that describe the surface that corresponds to a pixel. The shader that's described in this document utilizes the height of each pixel of a 3D object in world space to affect the final output color of the fragment.

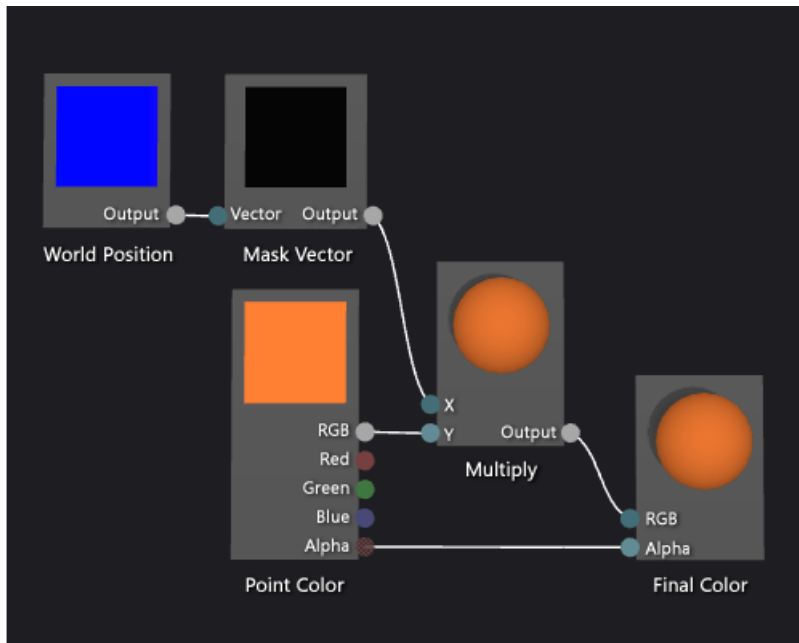
Before you begin, make sure that the **Properties** window and the **Toolbox** are displayed.

1. Create a DGSL shader with which to work. For information about how to add a DGSL shader to your project, see the Getting Started section in [Shader Designer](#).
2. Disconnect the **Point Color** node from the **Final Color** node. Choose the **RGB** terminal of the **Point Color** node, and then choose **Break Links**. This makes room for the node that's added in the next step.
3. Add a **Multiply** node to the graph. In the **Toolbox**, under **Math**, select **Multiply** and move it to the design surface.
4. Add a **Mask Vector** node to the graph. In the **Toolbox**, under **Utility**, select **Mask Vector** and move it to the design surface.
5. Specify mask values for the **Mask Vector** node. In **Select** mode, select the **Mask Vector** node, and then in the **Properties** window, set the **Green / Y** property to **True**, and then set the **Red / X**, **Blue / Z** and **Alpha / W** properties to **False**. In this example, the **Red / X**, **Green / Y**, and **Blue / Z** properties correspond to the x, y, and z components of the **World Position** node, and **Alpha / W** is unused. Because only **Green / Y** is set to **True**, only the y component of the input vector remains after it is masked.
6. Add a **World Position** node to the graph. In the **Toolbox**, under **Constants**, select **World Position** and move it to the design surface.
7. Mask the world space position of the fragment. In **Select** mode, move the **Output** terminal of the **World Position** node to the **Vector** terminal of the **Mask Vector** node. This connection masks the position of the fragment to ignore the x and z components.
8. Multiply the RGB color constant by the masked world space position. Move the **RGB** terminal of the **Point Color** node to the **Y** terminal of the **Multiply** node, and then move the **Output** terminal of the **Mask Vector** node to the **X** terminal of the **Multiply** node. This connection scales the color value by the height of the pixel in world space.
9. Connect the scaled color value to the final color. Move the **Output** terminal of the **Multiply** node to the **RGB** terminal of the **Final Color** node.

The following illustration shows the completed shader graph and a preview of the shader applied to a sphere.

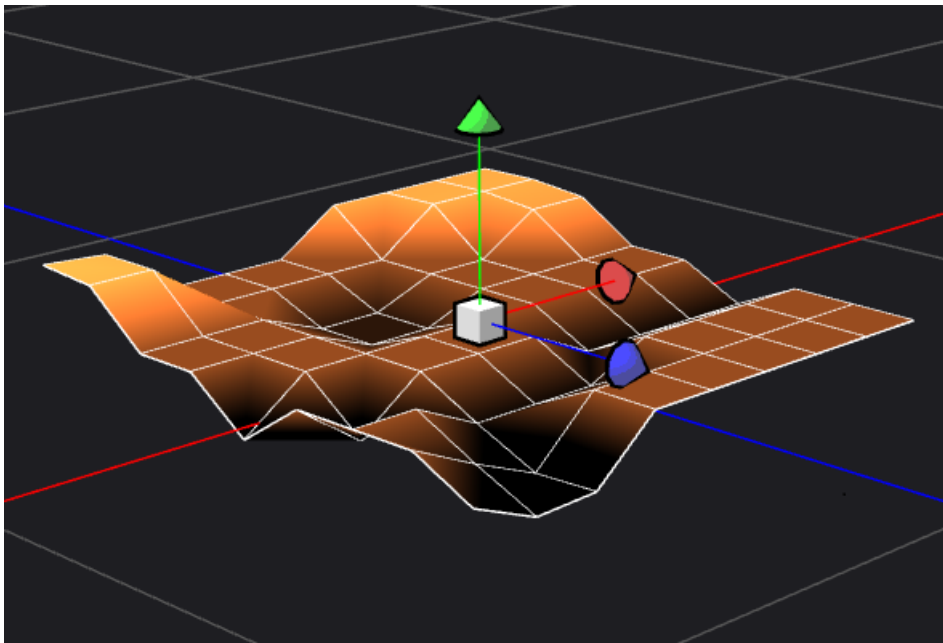
NOTE

In this illustration, an orange color is specified to better demonstrate the effect of the shader, but because the preview shape has no position in world-space, the shader cannot be fully previewed in the Shader Designer. The shader must be previewed in a real scene to demonstrate the full effect.



Certain shapes might provide better previews for some shaders. For information about how to preview shaders in the Shader Designer, see **Previewing shaders** in [Shader Designer](#)

The following illustration shows the shader that's described in this document applied to the 3D scene that's demonstrated in [How to: Model 3D terrain](#). The intensity of the color increases with the height of the point in the world.



For more information about how to apply a shader to a 3D model, see [How to: Apply a shader to a 3D model](#).

See also

- [How to: Apply a shader to a 3D model](#)
- [How to: Export a shader](#)

- [How to: Model 3D terrain](#)
- [How to: Create a grayscale texture shader](#)
- [Shader Designer](#)
- [Shader Designer nodes](#)

Walkthrough: Create a realistic 3D billiard ball

2/8/2019 • 13 minutes to read • [Edit Online](#)

This walkthrough demonstrates how to create a realistic 3D billiard ball by using the Shader Designer and Image Editor in Visual Studio. The 3D appearance of the billiard ball is achieved by combining several shader techniques with appropriate texture resources.

Prerequisites

You need the following components and skills to complete this walkthrough:

- A tool for assembling textures into a cube map, such as the DirectX Texture Tool that is included in the June 2010 DirectX SDK.
- Familiarity with the Image Editor in Visual Studio.
- Familiarity with the Shader Designer in Visual Studio.

Create the basic appearance with shape and texture

In computer graphics, the most-basic elements of appearance are shape and color. In a computer simulation, it is common to use a 3D model to represent the shape of a real-world object. Color detail is then applied to the surface of the model by using a texture map.

Typically, you might have to ask an artist to create a 3D model that you can use, but because a billiard ball is a common shape (a sphere), the Shader Designer already has a suitable model built in.

The sphere is the default preview shape in the Shader Designer; if you are currently using a different shape to preview your shader, switch back to the sphere.

To preview the shader by using a sphere

- On the Shader Designer toolbar, choose **Preview with sphere**.

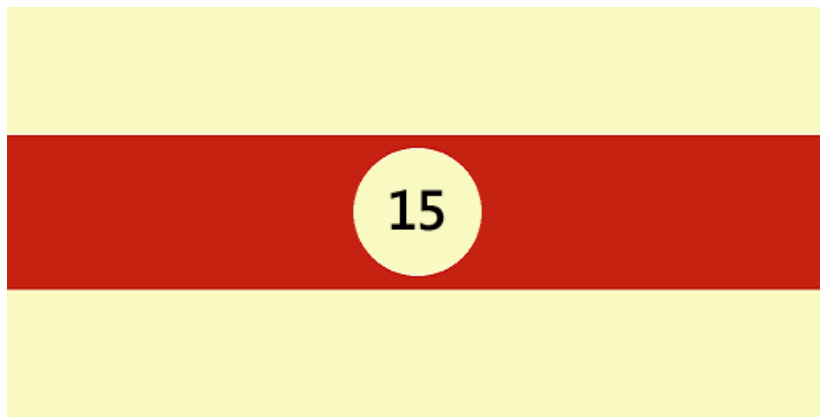
In the next step, you'll create a shader program that applies a texture to the model, but first you have to create a texture that you can use. This walkthrough demonstrates how to create the texture by using the Image Editor, which is a part of Visual Studio, but you can use any image editor that can save the texture in a suitable format.

Make sure that the **Properties** window and the **Toolbox** are displayed.

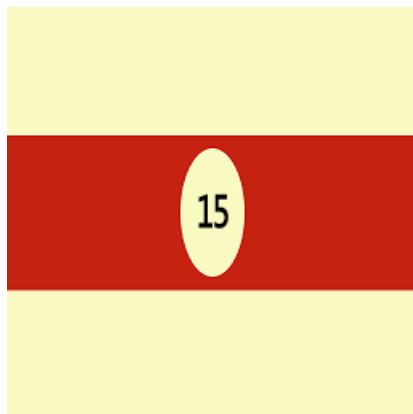
To create a billiard ball texture by using the Image Editor

1. Create a texture to work with. For information about how to add a texture to your project, see the Getting Started section in [Image Editor](#).
2. Set the image size so that its width is twice its height; this is necessary because of the way that a texture is mapped onto the spherical surface of the billiard ball. To resize the image, in the **Properties** window, specify new values for the **Width** and **Height** properties. For example, set the width to 512 and the height to 256.
3. Draw a texture for the billiard ball, keeping in mind how a texture is mapped onto a sphere.

The texture should look similar to this:



4. Optionally, you might want to decrease the storage requirements of this texture. You can do that by reducing the width of the texture to match its height. This compresses the texture along its width, but due to the way that the texture is mapped to the sphere, it will be expanded when the billiard ball is rendered. After resizing, the texture should look similar to this:

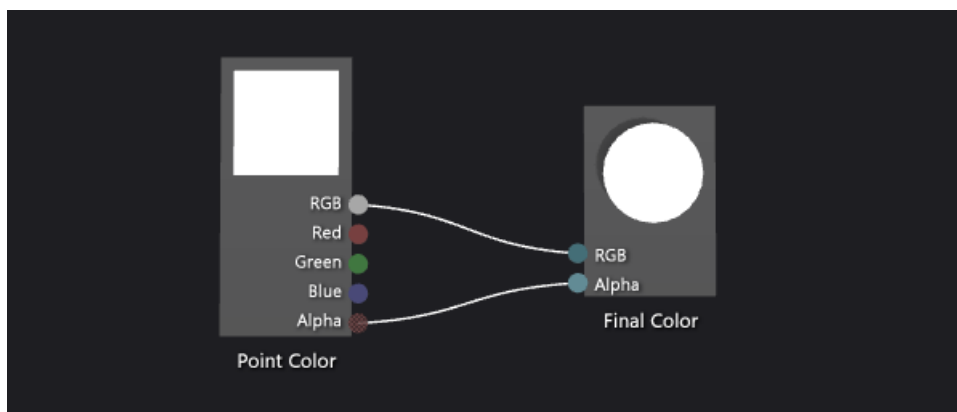


Now, you can create a shader that applies this texture to the model.

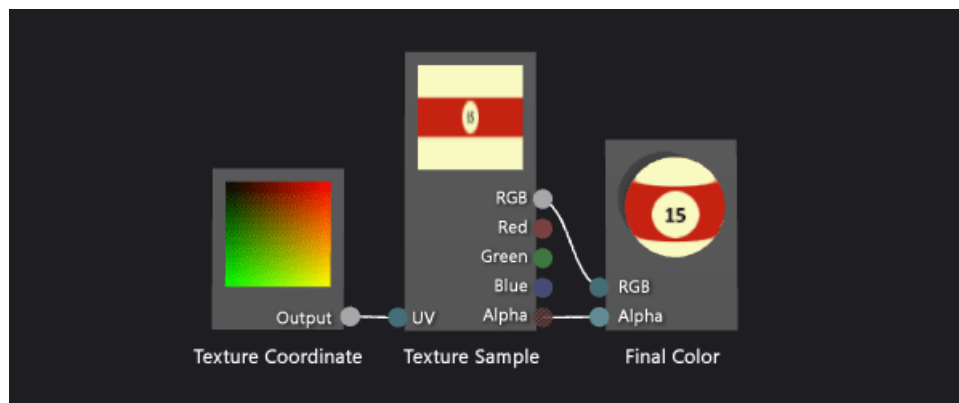
To create a basic texture shader

1. Create a DGSL shader with which to work. For information about how to add a DGSL shader to your project, see the Getting Started section in [Shader Designer](#).

By default, a shader graph looks like this:



2. Modify the default shader so that it applies the value of a texture sample to the current pixel. The shader graph should look like this:



3. Apply the texture that you created in the previous procedure by configuring the texture properties. Set the value of the **Texture** property of the **Texture Sample** node to **Texture1**, and then specify the texture file by using the **Filename** property of the **Texture1** property group in the same property window.

For more information about how to apply a texture in your shader, see [How to: Create a basic texture shader](#).

Your billiard ball should now look similar to this:



Create depth with the Lambert lighting model

So far, you've created an easily recognizable billiard ball. However, it appears flat and uninteresting—more like a cartoon picture of a billiard ball than a convincing replica. The flat appearance results from the simplistic shader, which behaves as if each pixel on the surface of the billiard ball receives the same amount of light.

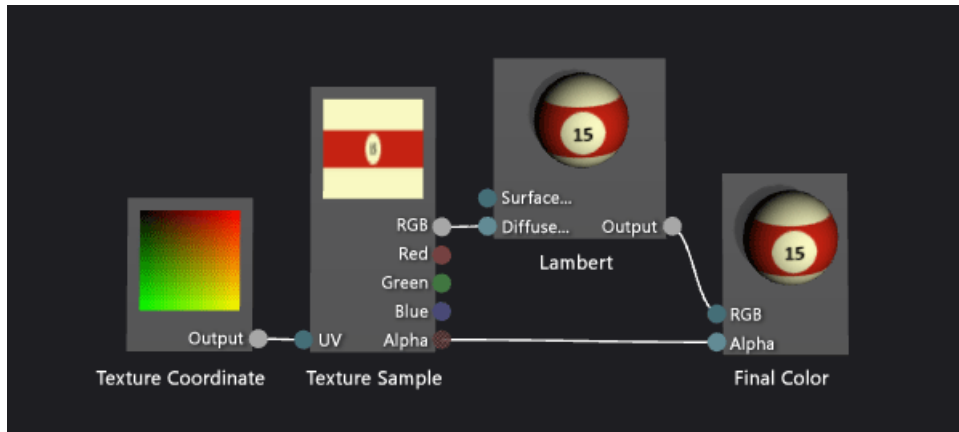
In the real world, light appears brightest on surfaces that directly face a light source and appears less bright on surfaces that are at an oblique angle to the light source. This is because the energy in the light rays is distributed across the smallest surface area when the surface directly faces the light source. As the surface turns away from the light source, the same amount of energy is distributed across an increasingly larger surface area. A surface that faces away from a light source receives no light energy at all, resulting in a completely dark appearance. This variance in brightness across the surface of an object is an important visual cue that helps indicate the shape of an object; without it, the object appears flat.

In computer graphics, *lighting models*—simplified approximations of complex, real-world lighting interactions—are used to replicate the appearance of real-world lighting. The Lambert lighting model varies the amount of diffusely reflected light across the surface of an object as described in the previous paragraph. You can add the

Lambert lighting model to your shader to give the billiard ball a more convincing 3D appearance.

To add Lambert lighting to your shader

- Modify your shader to modulate the value of the texture sample by the Lambert lighting value. Your shader graph should look like this:



- Optionally, you can adjust how the lighting behaves by configuring the **MaterialDiffuse** property of the shader graph. To access properties of the shader graph, choose an empty area of the design surface, and then locate the property that you want to access in the **Properties** window.

For more information about how to apply Lambert lighting in your shader, see [How to: Create a basic Lambert shader](#).

With Lambert lighting applied, your billiard ball should look similar to this:



Enhance the basic appearance with specular highlights

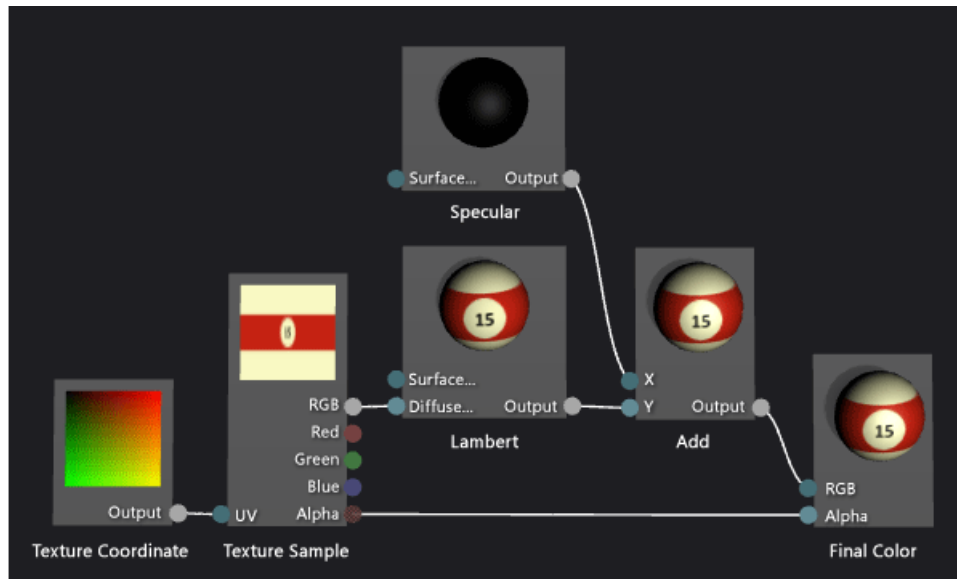
The Lambert lighting model provides the sense of shape and dimension that was absent in the texture-only shader. However, the billiard ball still has a somewhat dull appearance.

A real billiard ball usually has a glossy finish that reflects a portion of the light that falls on it. Some of this reflected light results in specular highlights, which simulate the reflecting properties of a surface. Depending on the properties of the finish, the highlights can be localized or broad, intense or subtle. These specular reflections are modeled by using the relationship between a light source, the orientation of the surface, and the camera position—that is, the highlight is most intense when the orientation of the surface reflects the light source directly into the camera, and is less intense when the reflection is less direct.

The Phong lighting model builds on the Lambert lighting model to include specular highlights as described in the previous paragraph. You can add the Phong lighting model to your shader to give the billiard ball a simulated finish that results in a more interesting appearance.

To add specular highlights to your shader

1. Modify your shader to include the specular contribution by using additive blending. Your shader graph should look like this:



2. Optionally, you can adjust the way that the specular highlight behaves by configuring the specular properties (**MaterialSpecular** and **MaterialSpecularPower**) of the shader graph. To access properties of the shader graph, choose an empty area of the design surface, and then in the **Properties** window, locate the property that you want to access.

For more information about how to apply specular highlights in your shader, see [How to: Create a basic Phong shader](#).

With specular highlighting applied, your billiard ball should look similar to this:



Create a sense of space by reflecting the environment

With specular highlights applied, your billiard ball looks pretty convincing. It's got the right shape, the right paint job, and the right finish. However, there's still one more technique that will make your billiard ball look more like a

part of its environment.

If you examine a real billiard ball closely, you can see that its glossy surface doesn't just exhibit specular highlights but also faintly reflects an image of the world around it. You can simulate this reflection by using an image of the environment as a texture and combining it with the model's own texture to determine the final color of each pixel. Depending on the kind of finish you want, you can combine more or less of the reflection texture together with the rest of the shader. For example, a shader that simulates a highly reflective surface like a mirror might use only the reflection texture, but a shader that simulates a more subtle reflection like the one found on a billiard ball might combine just a small portion of the reflection texture's value together with the rest of the shader calculation.

Of course, you can't just apply the reflected image to the model in the same way that you apply the model's texture map. If you did, the reflection of the world would move with the billiard ball as if the reflection were glued to it. Because a reflection can come from any direction, you need a way to provide a reflection map value for any angle, and a way to keep the reflection map oriented according to the world. To satisfy these requirements you can use a special kind of texture map—called a *cube map*—that provides six textures arranged to form the sides of a cube. From inside this cube, you can point in any direction to find a texture value. If the textures on each side of the cube contain images of the environment, you can simulate any reflection by sampling the correct location on the surface of the cube. By keeping the cube aligned to the world, you'll get an accurate reflection of the environment. To determine where the cube should be sampled, you just calculate the reflection of the camera vector off the surface of the object, and then use it as 3D texture coordinates. Using cube maps in this way is a common technique that's known as *environment mapping*.

Environment mapping provides an efficient approximation of real reflections as described in the preceding paragraphs. You can blend environment-mapped reflections into your shader to give the billiard ball a simulated finish that makes the billiard ball seem more grounded in the scene.

The first step is to create a cube map texture. In many kinds of apps, the contents of the cube map don't have to be perfect to be effective, especially when the reflection is subtle or doesn't occupy a prominent space on the screen. For example, many games use pre-computed cube maps for environment mapping and just use the one nearest to each reflective object, although this means that the reflection isn't correct. Even a rough approximation is often good enough for a convincing effect.

To create textures for an environment map by using the Image Editor

1. Create a texture to work with. For information about how to add a texture to your project, see the Getting Started section in [Image Editor](#).
2. Set the image size so that its width is equal to its height, and is a power of two in size; this is necessary because of the way that a cube map is indexed. To resize the image, in the **Properties** window, specify new values for the **Width** and **Height** properties. For example, set the value of the **Width** and **Height** properties to 256.
3. Use a solid color to fill the texture. This texture will be the bottom of the cube map, which corresponds to the surface of the billiard table. Keep the color you used in mind for the next texture.
4. Create a second texture that is the same size as the first. This texture will be repeated on the four sides of the cube map, which correspond to the surface and sides of a billiard table, and to the area around the billiard table. Make sure to draw the surface of the billiard table in this texture by using the same color as in the bottom texture. The texture should look similar to this:



Remember that a reflection map doesn't have to be photorealistic to be effective; for example, the cube map used to create the images in this article contains just four pockets instead of six.

5. Create a third texture that is the same size as the others. This texture will be the top of the cube map, which corresponds to the ceiling above the billiard table. To make this part of the reflection more interesting, you can draw an overhead light to reinforce the specular highlights that you added to the shader in the previous procedure. The texture should look similar to this:



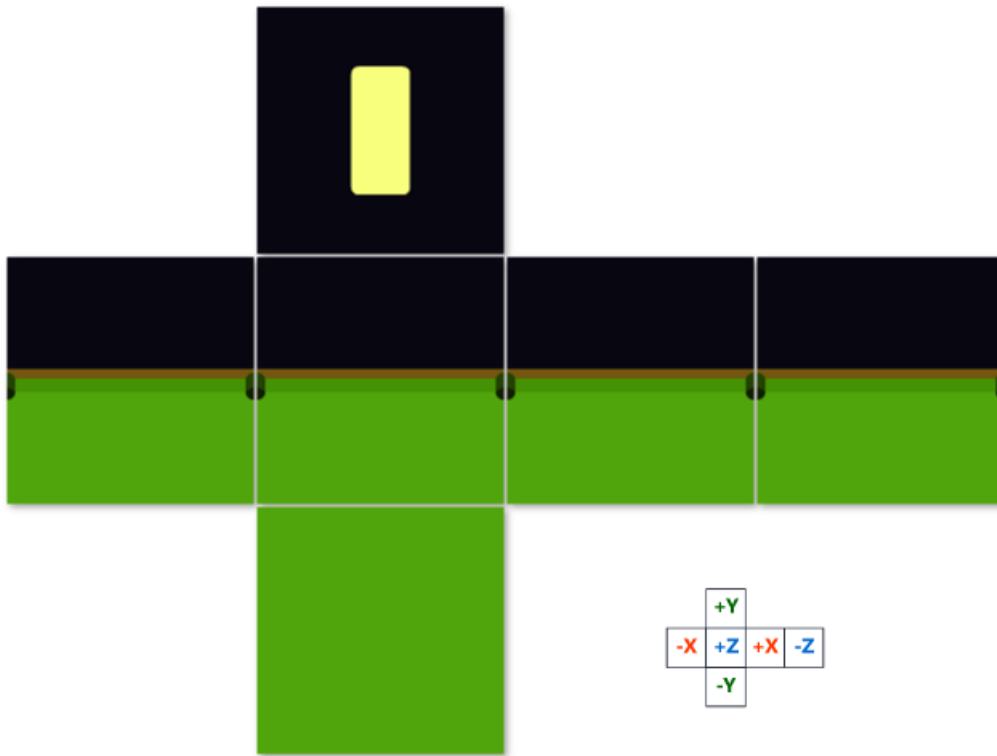
Now that you have created individual textures for the sides of the cube map, you can use a tool to assemble them into a cube map that can be stored in a single *.dds* texture. You can use any program you want to create the cube map as long as it can save the cube map in the *.dds* texture format. This walkthrough demonstrates how to create the texture by using the DirectX Texture Tool that's a part of the June, 2010 DirectX SDK.

To assemble a cube map by using the DirectX Texture Tool

1. In the DirectX Texture Tool, on the main menu, choose **File > New Texture**. The **New Texture** dialog box appears.
2. In the **Texture Type** group, choose **Cubemap Texture**.
3. In the **Dimensions** group, enter the correct value for the **Width** and **Height**, and then choose **OK**. A new texture document appears. By default, the texture first shown in the texture document corresponds to the **Positive X** cube face.
4. Load the texture that you created for the side of the texture cube onto the cube face. On the main menu, choose **File > Open Onto This Cubemap Face**, select the texture that you created for the side of the cube, and then choose **Open**.
5. Repeat step 4 for the **Negative X**, **Positive Z**, and **Negative Z** cube faces. To do so, you must view the face that you want to load. To view a different cube map face, on the main menu, choose **View > Cube Map Face**, and then select the face that you want to view.
6. For the **Positive Y** cube face, load the texture that you created for the top of the texture cube.

- For the **Negative Y** cube face, load the texture that you created for the bottom of the texture cube.
- Save the texture.

You can imagine the layout of the cube map like this:

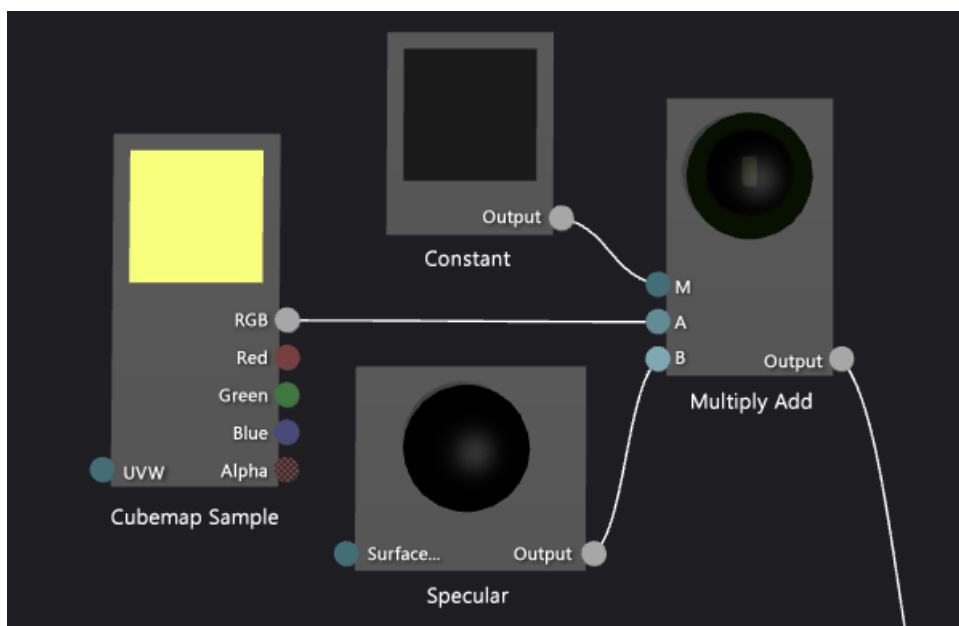


The image at the top is the positive Y (+Y) cube face; in the middle, from left to right, is the -X, +Z, +X, and -Z cube faces; at the bottom is the -Y cube face.

Now you can modify the shader to blend the cube map sample into the rest of the shader.

To add environment mapping to your shader

- Modify your shader to include the environment mapping contribution by using additive blending. Your shader graph should look like this:



Note that you can use a **Multiply-Add** node to simplify the shader graph.

Here's a more detailed view of the shader nodes that implement environment mapping:

2. Apply the texture that you created in the previous procedure by configuring the texture properties of the cube map. Set the value of the **Texture** property of the **Cubemap Sample** node to **Texture2**, and then specify the texture file by using the **Filename** property of the **Texture2** property group.
3. Optionally, you can adjust the reflectivity of the billiard ball by configuring the **Output** property of the **Constant** node. To access properties of the node, select it and then in the **Properties** window, locate the property that you want to access.

With environment mapping applied, your billiard ball should look similar to this:



In this final image, notice how the effects that you added come together to create a very convincing billiard ball. The shape, texture, and lighting create the basic appearance of a 3D object, and the specular highlights and reflections make the billiard ball more interesting and look like a part of its environment.

See also

- How to: Export a shader
- How to: Apply a shader to a 3D model

- [Shader Designer](#)
- [Image Editor](#)
- [Shader Designer nodes](#)

How to: Export a shader

2/8/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use the **Shader Designer** to export a Directed Graph Shader Language (DGSL) shader so that you can use it in your app.

Export a shader

After you create a shader by using the Shader Designer and before you can use it in your app, you have to export it in a format that your graphics API understands. You can export a shader in different ways to meet different needs.

1. In Visual Studio, open a **Visual Shader Graph (.dgsi)** file.

If you don't have a **Visual Shader Graph (.dgsi)** file to open, create one as described in [How to: Create a basic color shader](#).

2. On the **Shader Designer** toolbar, choose **Advanced** > **Export** > **Export As**. The **Export Shader** dialog box appears.
3. In the **Save as type** drop-down list, choose the format that you want to export.

Here are the formats that you can choose:

HLSL Pixel Shader (*.hlsl) Exports the shader as High Level Shader Language (HLSL) source code. This option makes it possible to modify the shader later, even after it's deployed in an app. This can make it easier to debug and patch the code based on end-user problems, but it also makes it easier for a user to modify your shader in unwanted ways—for example, to gain an unfair advantage in a competitive game. It also might increase the load time of the shader.

Compiled Pixel Shader (*.cso) Exports the shader as HLSL bytecode. This option makes it possible to modify the shader later, even after it's deployed in an app. This can make it easier to debug and patch the code based on end-user problems, but because the shader is pre-compiled, it does not incur extra runtime overhead when the shader is loaded by the app. Sufficiently skilled users can still modify the shader in unwanted ways, but compiling the shader makes this significantly more difficult.

C++ Header (*.h) Exports the shader as a C-style header that defines a byte array that contains HLSL bytecode. This option can make it more time-consuming to debug and patch the code based on end-user problems because the app must be recompiled to test the fix. However, because this option makes it difficult, though not impossible, to modify the shader after it's deployed in an app, it presents the most difficulty to a user who wants to modify the shader in unwanted ways.

4. In the **File name** combo box, specify a name for the exported shader, and then choose the **Save** button.

See also

- [How to: Create a basic color shader](#)
- [Shader Designer](#)

Use 3D assets in your game or app

2/8/2019 • 8 minutes to read • [Edit Online](#)

This article describes how you can use Visual Studio to process 3D assets and include them in your builds.

After you use the tools in Visual Studio to create 3D assets, the next step is to use them in your app. But, before you can use them, your assets have to be transformed into a format that DirectX can understand. To help you transform your assets, Visual Studio provides build customizations for each kind of asset that it can produce. To include the assets in your build, all you have to do is configure your project to use the build customizations, add the assets to your project, and configure the assets to use the correct build customization. After that, you can load the assets into your app and use them by creating and filling DirectX resources just like you would in any other DirectX app.

Configure your project

Before you can deploy your 3D assets as part of your build, Visual Studio has to know about the kinds of assets that you want to deploy. Visual Studio already knows about many common file types, but because only certain kinds of apps use 3D assets, Visual Studio doesn't assume that a project will build these kinds of files. You can tell Visual Studio that your app uses these kinds of assets by using the *build customizations*—files that tell Visual Studio how to process different types of files in a useful way—that are provided for each asset type. Because these customizations are applied on a per-project basis, all you have to do is add the appropriate customizations to your project.

To add the build customizations to your project

1. In **Solution Explorer**, open the shortcut menu for the project, and then choose **Build Dependencies > Build Customizations**. The **Visual C++ Build Customizations Files** dialog box appears.
2. Under **Available Build Customization Files**, select the check boxes that correspond to the asset types that you want to use in your project, as described in the following table:

ASSET TYPE	BUILD CUSTOMIZATION NAME
Textures and images	ImageContentTask(.targets, .props)
3D Models	MeshContentTask(.targets, .props)
Shaders	ShaderGraphContentTask(.targets, .props)

3. Choose the **OK** button.

Include assets in your build

Now that your project knows about the different kinds of 3D assets that you want to use, the next step is to tell it which files are 3D assets, and which kinds of assets they are.

To add an asset to your build

1. In **Solution Explorer**, in your project, open the shortcut menu of an asset, and then choose **Properties**. The asset's **Property Page** dialog box appears.
2. Make sure that the **Configuration** and **Platform** properties are set to the values to which you want your changes to apply.

- Under **Configuration Properties**, choose **General**, and then in the property grid, under **General**, set the **Item Type** property to the appropriate content pipeline item type. For example, for an image or texture file, choose **Image Content Pipeline**.

IMPORTANT

By default, Visual Studio assumes that many kinds of image files should be categorized by using the **Image** item type that's built into Visual Studio. Therefore, you have to change the **Item Type** property of each image that you want to be processed by the image content pipeline. Other types of content pipeline source files for 3D models and visual shader graphics default to the correct **Item Type**.

- Choose the **OK** button.

Following are the three content pipeline item types and their associated source and output file types.

ITEM TYPE	SOURCE FILE TYPES	OUTPUT FILE FORMAT
Image Content Pipeline	Portable Network Graphics (.png) JPEG (.jpg, .jpeg, .jpe, .jfif) Direct Draw Surface (.dds) Graphics Interchange Format (.gif) Bitmap (.bmp, .dib) Tagged Image File Format (.tif, .tiff) Targa (.tga)	DirectDraw Surface (.dds)
Mesh Content Pipeline	AutoDesk FBX Interchange File (.fbx) Collada DAE File (.dae) Wavefront OBJ File (.obj)	3D mesh file (.cmo)
Shader Content Pipeline	Visual Shader Graph (.dgsi)	Compiled Shader Output (.cso)

Configure asset content pipeline properties

You can set the content pipeline properties of each asset file so that it will be built in a specific way.

To configure content pipeline properties

- In **Solution Explorer**, in your project, open the shortcut menu for the asset file, and then choose **Properties**. The asset's **Property Page** dialog box appears.
- Make sure that the **Configuration** and **Platform** properties are set to the values that you want your changes to apply to.
- Under **Configuration Properties**, choose the content pipeline node—for example, **Image Content Pipeline** for texture and image assets—and then in the property grid, set the properties to the appropriate values. For example, to generate mipmaps for a texture asset at build time, set the **Generate Mips** property to **Yes**.
- Choose the **OK** button.

Image content pipeline configuration

When you use the image content pipeline tool to build a texture asset, you can compress the texture in various ways, indicate whether MIP levels should be generated at build time, and change the name of the output file.

PROPERTY	DESCRIPTION
Compress	<p>Specifies the compression type that's used for the output file.</p> <p>The available options are:</p> <ul style="list-style-type: none">- No Compression- BC1_UNORM compression- BC1_UNORM_SRGB compression- BC2_UNORM compression- BC2_UNORM_SRGB compression- BC3_UNORM compression- BC3_UNORM_SRGB compression- BC4_UNORM compression- BC4_SNORM compression- BC5_UNORM compression- BC5_SNORM compression- BC6H_UF16 compression- BC6H_SF16 compression- BC7_UNORM compression- BC7_UNORM_SRGB compression <p>For information about which compression formats are supported in different versions of DirectX, see Programming Guide for DXGI.</p>
Convert to pre-multiplied alpha format	Yes to convert the image to pre-multiplied alpha format in the output file; otherwise, No . Only the output file is changed, the source image is unchanged.
Generate Mips	Yes to generate a full MIP chain at build time and include it in the output file; otherwise, No . If No , and the source file already contains a mipmap chain, then the output file will have a MIP chain; otherwise, the output file will have no MIP chain.
Content Output	Specifies the name of the output file. Important: Changing the file name extension of the output file has no effect on its file format.

Mesh content pipeline configuration

When you use the mesh content pipeline tool to build a mesh asset, you can change the name of the output file.

PROPERTY	DESCRIPTION
Content Output	Specifies the name of the output file. Important: Changing the file name extension of the output file has no effect on its file format.

Shader content pipeline configuration

When you use the shader content pipeline tool to build a shader asset, you can change the name of the output file.

PROPERTY	DESCRIPTION
----------	-------------

PROPERTY	DESCRIPTION
Content Output	Specifies the name of the output file. Important: Changing the file name extension of the output file has no effect on its file format.

Load and use 3D assets at run time

Use textures and images

Direct3D provides functions for creating texture resources. In Direct3D 11, the D3DX11 utility library provides additional functions for creating texture resources and resource views directly from image files. For more information about how to create a texture resource in Direct3D 11, see [Textures](#). For more information about how to use the D3DX11 library to create a texture resource or resource view from an image file, see [How to: Initialize a texture from a file](#).

Use 3D models

Direct3D 11 does not provide functions for creating resources from 3D models. Instead, you have to write code that reads the 3D model file and creates vertex and index buffers that represent the 3D model and any resources that the model requires—for example, textures or shaders.

Use shaders

Direct3D provides functions for creating shader resources and binding them to the programmable graphics pipeline. For more information about how to create a shader resource in Direct3D and bind it to the pipeline, see [Programming guide for HLSL](#).

In the programmable graphics pipeline, each stage of the pipeline must give the next stage of the pipeline a result that's formatted in a way that it can understand. Because the Shader Designer can only create pixel shaders, this means that it's up to your app to ensure that the data that it receives is in the format that it expects. Several programmable shader stages occur before the pixel shader and perform geometric transformations—the vertex shader, the hull shader, the domain shader, and the geometry shader. The non-programmable tessellation stage also occurs before the pixel shader. No matter which of these stages directly precedes the pixel shader, it must give its result in this format:

```
struct PixelShaderInput
{
    float4 pos : SV_POSITION;
    float4 diffuse : COLOR;
    float2 uv : TEXCOORD0;
    float3 worldNorm : TEXCOORD1;
    float3 worldPos : TEXCOORD2;
    float3 toEye : TEXCOORD3;
    float4 tangent : TEXCOORD4;
    float3 normal : TEXCOORD5;
};
```

Depending on the Shader Designer nodes that you use in your shader, you might also have to provide additional data in the format according to these definitions:

```

Texture2D Texture1 : register( t0 );
Texture2D Texture2 : register( t1 );
Texture2D Texture3 : register( t2 );
Texture2D Texture4 : register( t3 );
Texture2D Texture5 : register( t4 );
Texture2D Texture6 : register( t5 );
Texture2D Texture7 : register( t6 );
Texture2D Texture8 : register( t7 );

TextureCube CubeTexture1 : register( t8 );
TextureCube CubeTexture2 : register( t9 );
TextureCube CubeTexture3 : register( t10 );
TextureCube CubeTexture4 : register( t11 );
TextureCube CubeTexture5 : register( t12 );
TextureCube CubeTexture6 : register( t13 );
TextureCube CubeTexture7 : register( t14 );
TextureCube CubeTexture8 : register( t15 );

SamplerState TexSampler : register( s0 );

cbuffer MaterialVars : register (b0)
{
    float4 MaterialAmbient;
    float4 MaterialDiffuse;
    float4 MaterialSpecular;
    float4 MaterialEmissive;
    float MaterialSpecularPower;
};

cbuffer LightVars : register (b1)
{
    float4 AmbientLight;
    float4 LightColor[4];
    float4 LightAttenuation[4];
    float3 LightDirection[4];
    float LightSpecularIntensity[4];
    uint IsPointLight[4];
    uint ActiveLights;
}

cbuffer ObjectVars : register(b2)
{
    float4x4 LocalToWorld4x4;
    float4x4 LocalToProjected4x4;
    float4x4 WorldToLocal4x4;
    float4x4 WorldToView4x4;
    float4x4 UVTransform4x4;
    float3 EyePosition;
};

cbuffer MiscVars : register(b3)
{
    float ViewportWidth;
    float ViewportHeight;
    float Time;
};

```

Related topics

TITLE	DESCRIPTION
How to: Export a texture that contains mipmaps	Describes how to use the Image Content Pipeline to export a texture that contains precomputed mipmaps.

TITLE	DESCRIPTION
How to: Export a texture that has premultiplied alpha	Describes how to use the Image Content Pipeline to export a texture that contains premultiplied alpha values.
How to: Export a texture for use with Direct2D or Javascript apps	Describes how to use the Image Content Pipeline to export a texture that can be used in a Direct2D or JavaScript app.
Working with 3D assets for games and apps	Describes the editing tools that Visual Studio provides for creating and manipulating 3D assets, which include textures and images, 3D models, and shaders.
How to: Export a shader	Describes how to export a shader from the Shader Designer.

How to: Export a texture that contains mipmaps

2/8/2019 • 2 minutes to read • [Edit Online](#)

The Image Content Pipeline can generate mipmaps from a source image as part of your project's build phase. To achieve certain effects, sometimes you have to specify the image content of each MIP level manually. When you don't need to specify the image content of each MIP level manually, generating mipmaps at build time ensures that mipmap contents never become out-of-sync. It also eliminates the performance cost of generating mipmaps at run time.

This article covers:

- Configuring the source image to be processed by the Image Content Pipeline.
- Configuring the Image Content Pipeline to generate mipmaps.

Export mipmaps

Mipmapping provides automatic screen-space Level-of-Detail for textured surfaces in a 3D game or app. It enhances the rendering performance of a game or app by pre-computing down-sampled versions of a texture. Pre-computing down-sampled versions means that the entire texture does not have to be down-sampled each time it's sampled.

To export a texture that has mipmaps

1. Begin with a basic texture. Load an existing image file, or create one as described in [How to: Create a basic texture](#). To support mipmaps, specify a texture that has a width and height that are both the same power of two in size, for example, 64x64, 256x256, or 512x512.
2. Configure the texture file you just created so that it's processed by the Image Content Pipeline. In **Solution Explorer**, open the shortcut menu for the texture file you created and then choose **Properties**. On the **Configuration Properties > General** page, set the **Item Type** property to **Image Content Pipeline**. Make sure that the **Content** property is set to **Yes** and **Exclude From Build** is set to **No**. Select **Apply**.

The **Image Content Pipeline** configuration property page appears.

3. Configure the Image Content Pipeline to generate mipmaps. On the **Configuration Properties > Image Content Pipeline > General** page, set the **Generate Mips** property to **Yes (/generatemips)**.
4. Select **OK**.

When you build the project, the Image Content Pipeline converts the source image from the working format to the output format that you specified, including MIP levels. The result is copied to the project's output directory.

How to: Export a texture that has premultiplied alpha

2/8/2019 • 2 minutes to read • [Edit Online](#)

The Image Content Pipeline can generate premultiplied alpha textures from a source image. These can be simpler to use and more robust than textures that do not contain premultiplied alpha.

This document demonstrates these activities:

- Configuring the source image to be processed by the Image Content Pipeline.
- Configuring the Image Content Pipeline to generate premultiplied alpha.

Premultiplied alpha

Premultiplied alpha offers several advantages over conventional, non-premultiplied alpha, because it better represents the real-world interaction of light with physical materials by separating the texel's color contribution (the color that it adds to the scene) from its translucency (the amount of underlying color that it allows through). Some of the advantages of using premultiplied alpha are:

- Blending with premultiplied alpha is an associative operation; the result of blending multiple translucent textures is the same, regardless of the order in which the textures are blended.
- Because of the associative nature of blending with premultiplied alpha, multi-pass rendering of translucent objects is simplified.
- By using premultiplied alpha, both pure additive blending (by setting alpha to zero) and linearly interpolated blending can be achieved simultaneously. For example, in a particle system, an additively blended fire particle can become a translucent smoke particle that's blended by using linear interpolation. Without premultiplied alpha, you would have to draw the fire particles separately from the smoke particles, and modify the render state between draw calls.
- Textures that use premultiplied alpha compress with higher quality than those that don't, and they don't exhibit the discolored edges—or "halo effect"—that can result when you blend textures that don't use premultiplied alpha.

To create a texture that uses premultiplied alpha

1. Begin with a basic texture. Load an existing image file, or create one as described in [How to: Create a basic texture](#).
2. Configure the texture file so that it's processed by the Image Content Pipeline. In **Solution Explorer**, open the shortcut menu for the texture file and then choose **Properties**. On the **Configuration Properties** > **General** page, set the **Item Type** property to **Image Content Pipeline**. Make sure that the **Content** property is set to **Yes** and **Exclude From Build** is set to **No**, and then choose the **Apply** button. The **Image Content Pipeline** configuration property page appears.
3. Configure the Image Content Pipeline to generate premultiplied alpha. On the **Configuration Properties** > **Image Content Pipeline** > **General** page, set the **Convert to pre-multiplied alpha format** property to **Yes (/generatepremultipliedalpha)**.
4. Choose the **OK** button.

When you build the project, the Image Content Pipeline converts the source image from the working format to the output format that you specified—this includes conversion of the image to premultiplied alpha format—and the result is copied to the project's output directory.

How to: Export a texture for use with Direct2D or Javascript apps

2/8/2019 • 2 minutes to read • [Edit Online](#)

The Image Content Pipeline can generate textures that are compatible with Direct2D's internal rendering conventions. Textures of this kind are suitable for use in apps that use Direct2D, and in UWP apps created by using JavaScript.

This document demonstrates these activities:

- Configuring the source image to be processed by the Image Content Pipeline.
- Configuring the Image Content Pipeline to generate a texture that you can use in a Direct2D or JavaScript app.
 - Generate a block-compressed *.dds* file.
 - Generate premultiplied alpha.
 - Disable mipmap generation.

Rendering conventions in Direct2D

Textures that are used in the context of Direct2D must conform to these Direct2D internal rendering conventions:

- Direct2D implements transparency and translucency by using premultiplied alpha. Textures used with Direct2D must contain premultiplied alpha, even if the texture doesn't use transparency or translucency. For more information about premultiplied alpha, see [How to: Export a texture that has Premultiplied Alpha](#).
- The texture must be supplied in *.dds* format, by using one of these block-compression formats:
 - BC1_UNORM compression
 - BC2_UNORM compression
 - BC3_UNORM compression
- Mipmaps are not supported.

To create a texture that's compatible with Direct2D rendering conventions

1. Begin with a basic texture. Load an existing image, or create a new one as described in [How to: Create a basic texture](#). To support block-compression in *.dds* format, specify a texture that has a width and height that are multiples of four in size, for example, 100x100, 128x128, or 256x192. Because mipmapping is not supported, the texture does not have to be square and does not have to be a power of two in size.
2. Configure the texture file so that it's processed by the Image Content Pipeline. In **Solution Explorer**, open the shortcut menu for the texture file you just created and then choose **Properties**. On the **Configuration Properties > General** page, set the **Item Type** property to **Image Content Pipeline**. Make sure that the **Content** property is set to **Yes** and **Exclude From Build** is set to **No**, and then choose the **Apply** button. The **Image Content Pipeline** configuration property page appears.
3. Set the output format to one of the block-compressed formats. On the **Configuration Properties > Image Content Pipeline > General** page, set the **Compress** property to **BC3_UNORM compression (/compress:BC3_UNORM)**. You could choose any of the other BC1, BC2, or BC3 formats, depending on

your requirements. Direct2D doesn't currently support BC4, BC5, BC6, or BC7 textures. For more information about the different BC formats, see [Block compression \(Direct3D 10\)](#).

NOTE

The compression format that's specified determines the format of the file that's produced by the Image Content Pipeline. This is different than the **Format** property of the source image in the Image Editor, which determines the format of the source image file as stored on disk—that is, the *working format*. Typically, you don't want a working format that's compressed.

4. Configure the Image Content Pipeline to produce output that uses premultiplied alpha. On the **Configuration Properties > Image Content Pipeline > General** page, set the **Convert to premultiplied alpha format** property to **Yes (/generatepremultipliedalpha)**.
5. Configure the image content pipeline so that it doesn't generate mipmaps. On the **Configuration Properties > Image Content Pipeline > General** page, set the **Generate Mips** property to **No**.
6. Choose the **OK** button.

When you build the project, the Image Content Pipeline converts the source image from the working format to the output format that you specified—conversion includes generation of premultiplied alpha—and the result is copied to the project's output directory.

The Visual Studio image library

2/8/2019 • 2 minutes to read • [Edit Online](#)

The Visual Studio Image Library contains application images that appear in Microsoft Visual Studio, Microsoft Windows, the Office system and other Microsoft software. This set of over 1,000 images can be used to create applications that look visually consistent with Microsoft software.

[Download the Visual Studio image library](#)

The image library is divided into five categories: Common Elements, Actions, Annotations, Icons, and Objects. Readme files are included in the PDF format for the Common Elements and Icon types. They contain information about how to use these images appropriately in your applications.

See also

- [Install Visual Studio](#)
- [Images, bitmaps, and metafiles](#)

Develop apps with the Workflow Designer

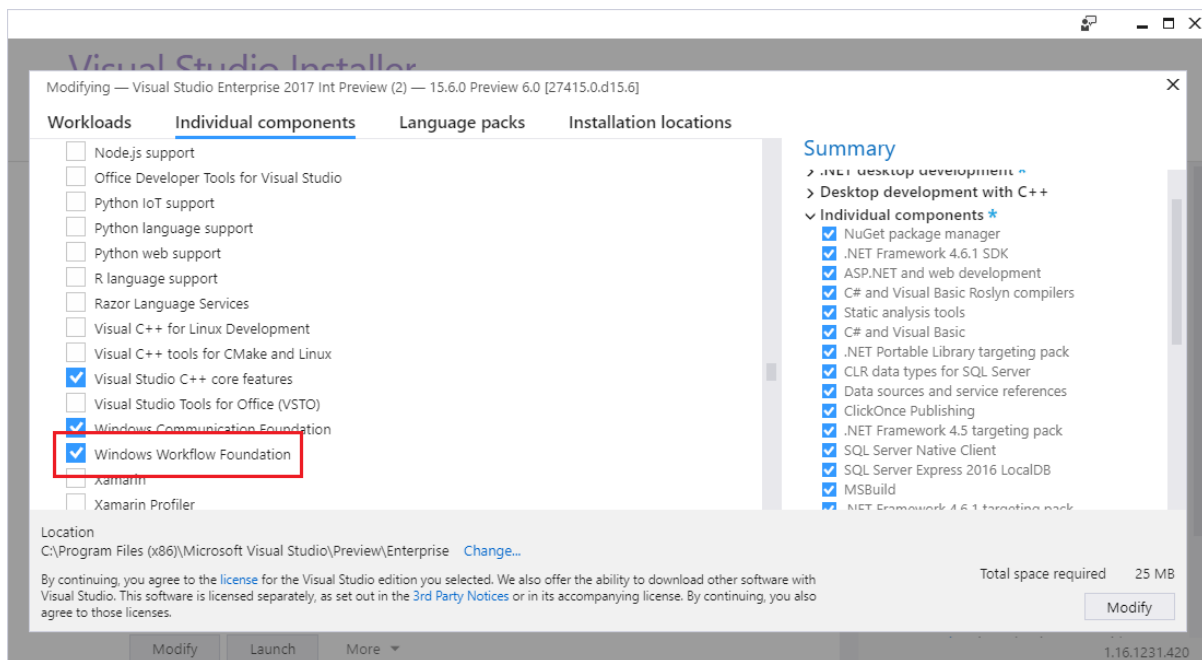
2/8/2019 • 2 minutes to read • [Edit Online](#)

The Workflow Designer is a visual designer and debugger for the graphical construction and debugging of [Windows Workflow Foundation](#) (WF) applications in Visual Studio. It enables you to compose a composite workflow application, activity library, or Windows Communication Foundation (WCF) service through the use of templates and activity designers.

Install Windows Workflow Foundation

To use Workflow project templates in Visual Studio 2017, first install the **Windows Workflow Foundation** component.

1. Open Visual Studio Installer. A quick way to open it is by selecting **Tools > Get Tools and Features** in Visual Studio.
2. In Visual Studio Installer, select the **Individual components** tab.
3. Scroll down to the **Development activities** category and then select the **Windows Workflow Foundation** component.



4. Select **Modify**.

Visual Studio installs the **Windows Workflow Foundation** component.

See also

- [Windows Workflow Foundation \(.NET Framework\)](#)