Painting with Solid Colors and Gradients Overview

Article • 03/17/2022 • 12 minutes to read • 1 contributor

In this article

This topic describes how to use SolidColorBrush, LinearGradientBrush, and RadialGradientBrush objects to paint with solid colors, linear gradients, and radial gradients.

Painting an Area with a Solid Color

One of the most common operations in any platform is to paint an area with a solid Color. To accomplish this task, Windows Presentation Foundation (WPF) provides the SolidColorBrush class. The following sections describe the different ways to paint with a SolidColorBrush.

Using a SolidColorBrush in "XAML"

To paint an area with a solid color in XAML, use one of the following options.

- Select a predefined solid color brush by name. For example, you can set a button's Background to "Red" or "MediumBlue". For a list of other predefined solid color brushes, see the static properties of the Brushes class. The following is an example.

XAML Copy

```xml
<!-- This button's background is painted with a red SolidColorBrush,
     described using a named color. -->
```

```
<Button Background="Red">A Button</Button>
```

- Choose a color from the 32-bit color palette by specifying the amounts of red, green, and blue to combine into a single solid color. The format for specifying a color from the 32-bit palette is "*#rrggbb*", where *rr* is a two digit hexadecimal number specifying the relative amount of red, *gg* specifies the amount of green, and *bb* specifies the amount of blue. Additionally, the color can be specified as "*#aarrggbb*" where *aa* specifies the *alpha* value, or transparency, of the color. This approach enables you to create colors that are partially transparent. In the following example, the Background of a Button is set to fully-opaque red using hexadecimal notation.

XAML Copy

```xml
<!-- This button's background is painted with a red SolidColorBrush,
     described using hexadecimal notation. -->
<Button Background="#FFFF0000">A Button</Button>
```

- Use property tag syntax to describe a SolidColorBrush. This syntax is more verbose but enables you to specify additional settings, such as the brush's opacity. In the following example, the Background properties of two Button elements are set to fully-opaque red. The first brush's color is described using a predefined color name. The second brush's color is described using hexadecimal notation.

XAML Copy

```xml
<!-- Both of these buttons' backgrounds are painted with red
     SolidColorBrush objects, described using object element
     syntax. -->
<Button>A Button

  <Button.Background>
    <SolidColorBrush Color="Red" />
  </Button.Background>
</Button>

<Button>A Button

  <Button.Background>
    <SolidColorBrush Color="#FFFF0000" />
  </Button.Background>
</Button>
```

Painting with a SolidColorBrush in Code

To paint an area with a solid color in code, use one of the following options.

- Use one of the predefined brushes provided by the Brushes class. In the following example, the Background of a Button is set to Red.

  C# Copy

  ```
  Button myButton = new Button();
  myButton.Content = "A Button";
  myButton.Background = Brushes.Red;
  ```

- Create a SolidColorBrush and set its Color property using a Color structure. You can use a predefined color from the Colors class or you can create a Color using the static FromArgb method.

  The following example shows how to set the Color property of a SolidColorBrush using a predefined color.

  C# Copy

  ```
  Button myButton = new Button();
  myButton.Content = "A Button";

  SolidColorBrush mySolidColorBrush = new SolidColorBrush();
  mySolidColorBrush.Color = Colors.Red;
  myButton.Background = mySolidColorBrush;
  ```

The static FromArgb enables you to specify the color's alpha, red, green, and blue values. The typical range for each of these values is 0-255. For example, an alpha value of 0 indicates that a color is completely transparent, while a value of 255 indicates the color is completely opaque. Likewise, a red value of 0 indicates that a color has no red in it, while a value of 255 indicates a color has the maximum amount of red possible. In the following example, a brush's color is described by specifying alpha, red, green, and blue values.

C# Copy

```
Button myButton = new Button();
myButton.Content = "A Button";

SolidColorBrush mySolidColorBrush = new SolidColorBrush();
mySolidColorBrush.Color =
    Color.FromArgb(
        255, // Specifies the transparency of the color.
        255, // Specifies the amount of red.
        0, // specifies the amount of green.
        0); // Specifies the amount of blue.
```

```
myButton.Background = mySolidColorBrush;
```

For additional ways to specify color, see the Color reference topic.

Painting an Area with a Gradient

A gradient brush paints an area with multiple colors that blend into each other along an axis. You can use them to create impressions of light and shadow, giving your controls a three-dimensional feel. You can also use them to simulate glass, chrome, water, and other smooth surfaces. WPF provides two types of gradient brushes: LinearGradientBrush and RadialGradientBrush.

Linear Gradients

A LinearGradientBrush paints an area with a gradient defined along a line, the *gradient axis*. You specify the gradient's colors and their location along the gradient axis using GradientStop objects. You may also modify the gradient axis, which enables you to create horizontal and vertical gradients and to reverse the gradient direction. The gradient axis is described in the next section. By default, a diagonal gradient is created.

The following example shows the code that creates a linear gradient with four colors.

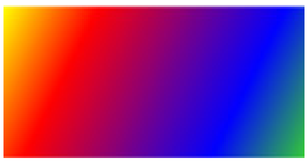
XAML Copy

```
<!-- This rectangle is painted with a diagonal linear gradient. -->
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

C# Copy

```
Rectangle diagonalFillRectangle = new Rectangle();
diagonalFillRectangle.Width = 200;
diagonalFillRectangle.Height = 100;

// Create a diagonal linear gradient with four stops.
LinearGradientBrush myLinearGradientBrush =
    new LinearGradientBrush();
myLinearGradientBrush.StartPoint = new Point(0,0);
myLinearGradientBrush.EndPoint = new Point(1,1);
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

// Use the brush to paint the rectangle.
diagonalFillRectangle.Fill = myLinearGradientBrush;
```

This code produces the following gradient:



ⓘ **Note**

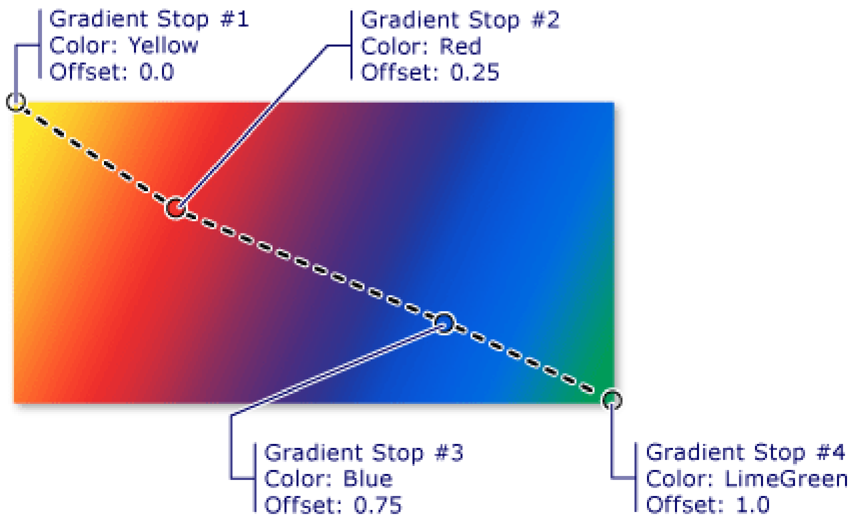
The gradient examples in this topic use the default coordinate system for setting start points and end points. The default coordinate system is relative to a bounding box: 0 indicates 0 percent of the bounding box and 1 indicates 100 percent of the bounding box. You can change this coordinate system by setting the **MappingMode** property to the value **Absolute**. An absolute coordinate system is not relative to a bounding box. Values are interpreted directly in local space.

The GradientStop is the basic building block of a gradient brush. A gradient stop specifies a Color at an Offset along the gradient axis.

- The gradient stop's Color property specifies the color of the gradient stop. You may set the color by using a predefined color (provided by the Colors class) or by specifying ScRGB or ARGB values. In XAML, you may also use hexadecimal notation to describe a color. For more information, see the Color structure.

- The gradient stop's Offset property specifies the position of the gradient stop's color on the gradient axis. The offset is a Double that ranges from 0 to 1. The closer a gradient stop's offset value is to 0, the closer the color is to the start of the gradient. The closer the gradient's offset value is to 1, the closer the color is to the end of the gradient.

The color of each point between gradient stops is linearly interpolated as a combination of the color specified by the two bounding gradient stops. The following illustration highlights the gradient stops in the previous example. The circles mark the position of gradient stops and a dashed line shows the gradient axis.

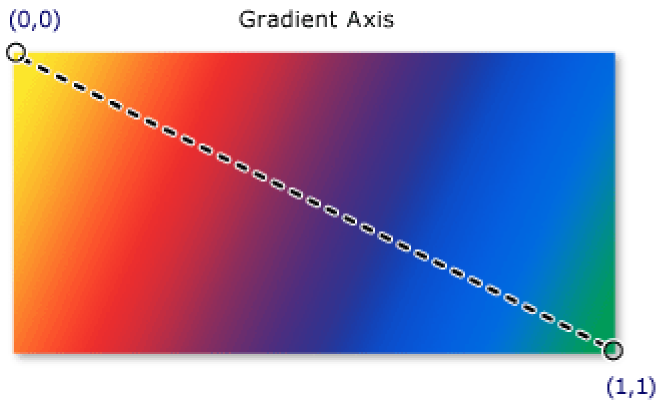


The first gradient stop specifies the color yellow at an offset of `0.0`. The second gradient stop specifies the color red at an offset of `0.25`. The points between these two stops gradually change from yellow to red as you move from left to right along the gradient axis. The third gradient stop specifies the color blue at an offset of `0.75`. The points between the second and third gradient stops gradually change from red to blue. The fourth gradient stop specifies the color lime green at an offset of `1.0`. The points between the third and fourth gradient stops gradually change from blue to lime green.

The Gradient Axis

As previously mentioned, a linear gradient brush's gradient stops are positioned along a line, the gradient axis. You may change the orientation and size of the line using the brush's StartPoint and EndPoint properties. By manipulating the brush's StartPoint and EndPoint, you can create horizontal and vertical gradients, reverse the gradient direction, condense the gradient spread, and more.

By default, the linear gradient brush's StartPoint and EndPoint are relative to the area being painted. The point (0,0) represents the upper-left corner of the area being painted, and (1,1) represents the lower-right corner of the area being painted. The default StartPoint of a LinearGradientBrush is (0,0), and its default EndPoint is (1,1), which creates a diagonal gradient starting at the upper-left corner and extending to the lower-right corner of the area being painted. The following illustration shows the gradient axis of a linear gradient brush with default StartPoint and EndPoint.



The following example shows how to create a horizontal gradient by specifying the brush's StartPoint and EndPoint. Notice that the gradient stops are the same as in the previous examples; by simply changing the StartPoint and EndPoint, the gradient has been changed from diagonal to horizontal.

XAML

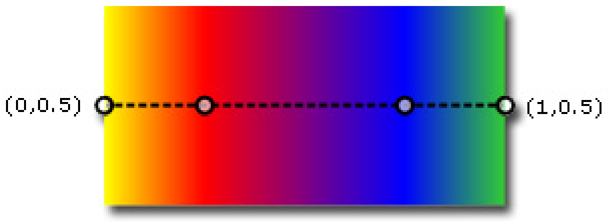
```xml
<!-- This rectangle is painted with a horizontal linear gradient. -->
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

C#

```csharp
Rectangle horizontalFillRectangle = new Rectangle();
horizontalFillRectangle.Width = 200;
horizontalFillRectangle.Height = 100;

// Create a horizontal linear gradient with four stops.
LinearGradientBrush myHorizontalGradient =
    new LinearGradientBrush();
myHorizontalGradient.StartPoint = new Point(0,0.5);
myHorizontalGradient.EndPoint = new Point(1,0.5);
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

// Use the brush to paint the rectangle.
```

```
horizontalFillRectangle.Fill = myHorizontalGradient;
```

The following illustration shows the gradient that is created. The gradient axis is marked with a dashed line, and the gradient stops are marked with circles.



The next example shows how to create a vertical gradient.

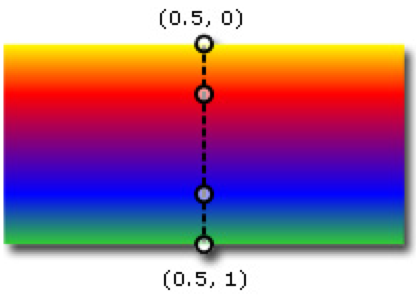
XAML Copy

```xml
<!-- This rectangle is painted with a vertical gradient. -->
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

C# Copy

```csharp
Rectangle verticalFillRectangle = new Rectangle();
verticalFillRectangle.Width = 200;
verticalFillRectangle.Height = 100;

// Create a vertical linear gradient with four stops.
LinearGradientBrush myVerticalGradient =
    new LinearGradientBrush();
myVerticalGradient.StartPoint = new Point(0.5,0);
myVerticalGradient.EndPoint = new Point(0.5,1);
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

// Use the brush to paint the rectangle.
verticalFillRectangle.Fill = myVerticalGradient;
```

The following illustration shows the gradient that is created. The gradient axis is marked with a dashed line, and the gradient stops are marked with circles.



Radial Gradients

Like a LinearGradientBrush, a RadialGradientBrush paints an area with colors that blend together along an axis. The previous examples showed how a linear gradient brush's axis is a straight line. A radial gradient brush's axis is defined by a circle; its colors "radiate" outward from its origin.

In the following example, a radial gradient brush is used to paint the interior of a rectangle.

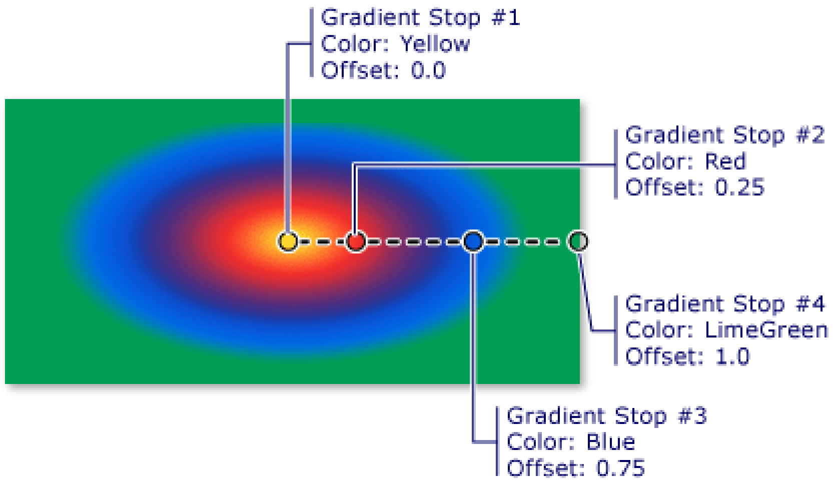
XAML Copy

```xml
<!-- This rectangle is painted with a diagonal linear gradient. -->
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <RadialGradientBrush
      GradientOrigin="0.5,0.5" Center="0.5,0.5"
      RadiusX="0.5" RadiusY="0.5">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1" />
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

C# Copy

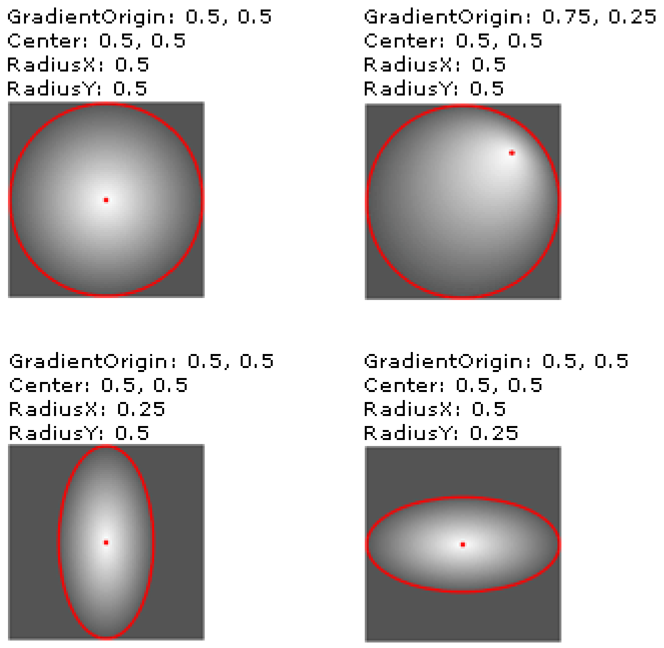
```csharp
RadialGradientBrush myRadialGradientBrush = new RadialGradientBrush();
myRadialGradientBrush.GradientOrigin = new Point(0.5,0.5);
myRadialGradientBrush.Center = new Point(0.5,0.5);
myRadialGradientBrush.RadiusX = 0.5;
myRadialGradientBrush.RadiusY = 0.5;
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

Rectangle myRectangle = new Rectangle();
myRectangle.Width = 200;
myRectangle.Height = 100;
myRectangle.Fill = myRadialGradientBrush;
```

The following illustration shows the gradient created in the previous example. The brush's gradient stops have been highlighted. Notice that, even though the results are different, the gradient stops in this example are identical to the gradient stops in the previous linear gradient brush examples.



The GradientOrigin specifies the start point of a radial gradient brush's gradient axis. The gradient axis radiates from the gradient origin to the gradient circle. A brush's gradient circle is defined by its Center, RadiusX, and RadiusY properties.

The following illustration shows several radial gradients with different GradientOrigin, Center, RadiusX, and RadiusY settings.



RadialGradientBrushes with different GradientOrigin, Center, RadiusX, and RadiusY settings.

Specifying Transparent or Partially-Transparent Gradient Stops

Because gradient stops do not provide an opacity property, you must specify the alpha channel of colors using ARGB hexadecimal notation in markup or use the Color.FromScRgb method to create gradient stops that are transparent or partially transparent. The following sections explain how to create partially transparent gradient stops in XAML and code.

Specifying Color Opacity in "XAML"

In XAML, you use ARGB hexadecimal notation to specify the opacity of individual colors. ARGB hexadecimal notation uses the following syntax:

`#` **aa** *rrggbb*

The *aa* in the previous line represents a two-digit hexadecimal value used to specify the opacity of the color. The *rr*, *gg*, and *bb* each represent a two digit hexadecimal value used to specify the amount of red, green, and blue in the color. Each hexadecimal digit may have a value from 0-9 or A-F. 0 is the smallest value, and F is the greatest. An alpha value of 00 specifies a color that is completely transparent, while an alpha value of FF creates a color that is fully opaque. In the following example, hexadecimal ARGB notation is used to specify two colors. The first is partially transparent (it has an alpha value of x20), while the second is completely opaque.

XAML Copy

```xml
<Rectangle Width="100" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0">

      <!-- This gradient stop is partially transparent. -->
      <GradientStop Color="#200000FF" Offset="0.0" />

      <!-- This gradient stop is fully opaque. -->
      <GradientStop Color="#FF0000FF" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Specifying Color Opacity in Code

When using code, the static FromArgb method enables you to specify an alpha value when you create a color. The method takes four parameters of type Byte. The first parameter specifies the alpha channel of the color; the other three parameters specify the red, green, and blue values of the color. Each value should be between 0 to 255, inclusive. An alpha value of 0 specifies that the color is completely transparent, while an alpha value of 255 specifies that the color is completely opaque. In the following example, the FromArgb method is used to produce two colors. The first color is partially transparent (it has an alpha value of 32), while the second is fully opaque.

C# Copy

```csharp
LinearGradientBrush myLinearGradientBrush = new LinearGradientBrush();

// This gradient stop is partially transparent.
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Color.FromArgb(32, 0, 0, 255), 0.0));

// This gradient stop is fully opaque.
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Color.FromArgb(255, 0, 0, 255), 1.0));

Rectangle myRectangle = new Rectangle();
myRectangle.Width = 100;
myRectangle.Height = 100;
myRectangle.Fill = myLinearGradientBrush;
```

Alternatively, you may use the FromScRgb method, which enables you to use ScRGB values to create a color.

Painting with Images, Drawings, Visuals, and Patterns

ImageBrush, DrawingBrush, and VisualBrush classes enable you to paint an area with images, drawings, or visuals. For information about painting with images, drawings, and patterns, see Painting with Images, Drawings, and Visuals.

See also

- Brush
- SolidColorBrush
- LinearGradientBrush
- RadialGradientBrush
- Painting with Images, Drawings, and Visuals
- Brush Transformation Overview
- Graphics Rendering Tiers

Recommended content

SolidColorBrush Class (System.Windows.Media)

Paints an area with a solid color.

How to: Create Text with a Shadow - WPF .NET Framework

Slider Styles and Templates - WPF .NET Framework

How to: Draw a Closed Shape by Using the Polygon Element - WPF .NET Framework

Control.Background Property (System.Windows.Controls)

Gets or sets a brush that describes the background of a control.

Animation Overview - WPF .NET Framework

Make an attractive user interface even more spectacular with dramatic screen transitions or vivid visual cues in Windows Presentation Foundation (WPF).

TextBox Styles and Templates - WPF .NET Framework

Learn about styles and templates for the Windows Presentation Foundation TextBox control. Modify the ControlTemplate to give the control a unique appearance.

Show more