How to: Use a Pen to Draw Lines

.NET Framework (current version)

To draw lines, you need a Graphics object and a Pen object. The Graphics object provides the DrawLine method, and the Pen object stores features of the line, such as color and width.

Example

The following example draws a line from (20, 10) to (300, 100). The first statement uses the Pen class constructor to create a black pen. The one argument passed to the Pen constructor is a Color object created with the FromArgb method. The values used to create the Color object — (255, 0, 0, 0) — correspond to the alpha, red, green, and blue components of the color. These values define an opaque black pen.

VB

```
Dim pen As New Pen(Color.FromArgb(255, 0, 0, 0))
e.Graphics.DrawLine(pen, 20, 10, 300, 100)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Pen
Using a Pen to Draw Lines and Shapes
Pens, Lines, and Rectangles in GDI+

How to: Use a Pen to Draw Rectangles

.NET Framework (current version)

To draw rectangles, you need a Graphics object and a Pen object. The Graphics object provides the DrawRectangle method, and the Pen object stores features of the line, such as color and width.

Example

The following example draws a rectangle with its upper-left corner at (10, 10). The rectangle has a width of 100 and a height of 50. The second argument passed to the Pen constructor indicates that the pen width is 5 pixels.

When the rectangle is drawn, the pen is centered on the rectangle's boundary. Because the pen width is 5, the sides of the rectangle are drawn 5 pixels wide, such that 1 pixel is drawn on the boundary itself, 2 pixels are drawn on the inside, and 2 pixels are drawn on the outside. For more details on pen alignment, see How to: Set Pen Width and Alignment.

The following illustration shows the resulting rectangle. The dotted lines show where the rectangle would have been drawn if the pen width had been one pixel. The enlarged view of the upper-left corner of the rectangle shows that the thick black lines are centered on those dotted lines.

VB

```
Dim blackPen As New Pen(Color.FromArgb(255, 0, 0, 0), 5)
e.Graphics.DrawRectangle(blackPen, 10, 10, 100, 50)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Pen to Draw Lines and Shapes

How to: Set Pen Width and Alignment

.NET Framework (current version)

When you create a Pen, you can supply the pen width as one of the arguments to the constructor. You can also change the pen width with the Width property of the Pen class.

A theoretical line has a width of 0. When you draw a line that is 1 pixel wide, the pixels are centered on the theoretical line. If you draw a line that is more than one pixel wide, the pixels are either centered on the theoretical line or appear to one side of the theoretical line. You can set the pen alignment property of a Pen to determine how the pixels drawn with that pen will be positioned relative to theoretical lines.

The values Center, Outset, and Inset that appear in the following code examples are members of the PenAlignment enumeration.

The following code example draws a line twice: once with a black pen of width 1 and once with a green pen of width 10.

To vary the width of a pen

- Set the value of the Alignment property to Center (the default) to specify that pixels drawn with the green pen will be centered on the theoretical line. The following illustration shows the resulting line.

The following code example draws a rectangle twice: once with a black pen of width 1 and once with a green pen of width 10.

VB

```
Dim blackPen As New Pen(Color.FromArgb(255, 0, 0, 0), 1)
Dim greenPen As New Pen(Color.FromArgb(255, 0, 255, 0), 10)
greenPen.Alignment = PenAlignment.Center

' Draw the line with the wide green pen.
e.Graphics.DrawLine(greenPen, 10, 100, 100, 50)

' Draw the line with the thin black pen.
e.Graphics.DrawLine(blackPen, 10, 100, 100, 50)
```

To change the alignment of a pen

- Set the value of the Alignment property to Center to specify that the pixels drawn with the green pen will be centered on the boundary of the rectangle.

The following illustration shows the resulting rectangle.

VB

```
Dim blackPen As New Pen(Color.FromArgb(255, 0, 0, 0), 1)
Dim greenPen As New Pen(Color.FromArgb(255, 0, 255, 0), 10)
greenPen.Alignment = PenAlignment.Center

' Draw the rectangle with the wide green pen.
e.Graphics.DrawRectangle(greenPen, 10, 100, 50, 50)

' Draw the rectangle with the thin black pen.
e.Graphics.DrawRectangle(blackPen, 10, 100, 50, 50)
```

To create an inset pen

- Change the green pen's alignment by modifying the third statement in the preceding code example as follows:

VB

```
greenPen.Alignment = PenAlignment.Inset
```

Now the pixels in the wide green line appear on the inside of the rectangle as shown in the following illustration.

See Also

Using a Pen to Draw Lines and Shapes
Graphics and Drawing in Windows Forms

How to: Draw a Line with Line Caps

.NET Framework (current version)

You can draw the start or end of a line in one of several shapes called line caps. GDI+ supports several line caps, such as round, square, diamond, and arrowhead.

Example

You can specify line caps for the start of a line (start cap), the end of a line (end cap), or the dashes of a dashed line (dash cap).

The following example draws a line with an arrowhead at one end and a round cap at the other end. The illustration shows the resulting line:

VB

```
Dim pen As New Pen(Color.FromArgb(255, 0, 0, 255), 8)
pen.StartCap = LineCap.ArrowAnchor
pen.EndCap = LineCap.RoundAnchor
e.Graphics.DrawLine(pen, 20, 175, 300, 175)
```

Compiling the Code

- Create a Windows Form and handle the form's Paint event. Paste the example code into the Paint event handler passing *e* as PaintEventArgs.

See Also

System.Drawing.Pen
System.Drawing.Drawing2D.LineCap
Graphics and Drawing in Windows Forms
Using a Pen to Draw Lines and Shapes

How to: Join Lines

.NET Framework (current version)

A line join is the common area that is formed by two lines whose ends meet or overlap. GDI+ provides three line join styles: miter, bevel, and round. Line join style is a property of the Pen class. When you specify a line join style for a Pen object, that join style will be applied to all the connected lines in any GraphicsPath object drawn using that pen.

The following illustration shows the results of the beveled line join example.

Example

You can specify the line join style by using the LineJoin property of the Pen class. The example demonstrates a beveled line join between a horizontal line and a vertical line. In the following code, the value Bevel assigned to the LineJoin property is a member of the LineJoin enumeration. The other members of the LineJoin enumeration are Miter and Round.

VB

```
Dim path As New GraphicsPath()
Dim penJoin As New Pen(Color.FromArgb(255, 0, 0, 255), 8)

path.StartFigure()
path.AddLine(New Point(50, 200), New Point(100, 200))
path.AddLine(New Point(100, 200), New Point(100, 250))

penJoin.LineJoin = LineJoin.Bevel
e.Graphics.DrawPath(penJoin, path)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Pen to Draw Lines and Shapes

How to: Draw a Custom Dashed Line

.NET Framework (current version)

GDI+ provides several dash styles that are listed in the DashStyle enumeration. If those standard dash styles do not suit your needs, you can create a custom dash pattern.

Example

To draw a custom dashed line, put the lengths of the dashes and spaces in an array and assign the array as the value of the DashPattern property of a Pen object. The following example draws a custom dashed line based on the array `{5, 2, 15, 4}`. If you multiply the elements of the array by the pen width of 5, you get `{25, 10, 75, 20}`. The displayed dashes alternate in length between 25 and 75, and the spaces alternate in length between 10 and 20.

The following illustration shows the resulting dashed line. Note that the final dash has to be shorter than 25 units so that the line can end at (405, 5).

VB

```
Dim dashValues As Single() = {5, 2, 15, 4}
Dim blackPen As New Pen(Color.Black, 5)
blackPen.DashPattern = dashValues
e.Graphics.DrawLine(blackPen, New Point(5, 5), New Point(405, 5))
```

Compiling the Code

Create a Windows Form and handle the form's Paint event. Paste the preceding code into the Paint event handler.

See Also

Using a Pen to Draw Lines and Shapes

How to: Draw a Line Filled with a Texture

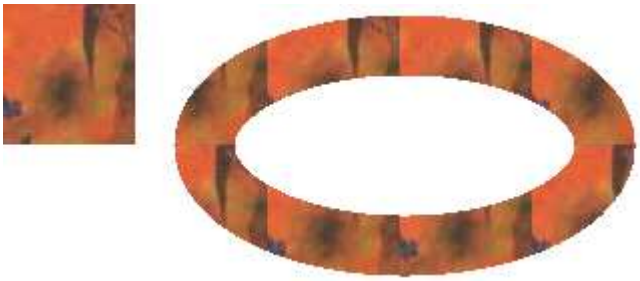
.NET Framework (current version)

Instead of drawing a line with a solid color, you can draw a line with a texture. To draw lines and curves with a texture, create a TextureBrush object, and pass that TextureBrush object to a Pen constructor. The bitmap associated with the texture brush is used to tile the plane (invisibly), and when the pen draws a line or curve, the stroke of the pen uncovers certain pixels of the tiled texture.

Example

The following example creates a Bitmap object from the file `Texture1.jpg`. That bitmap is used to construct a TextureBrush object, and the TextureBrush object is used to construct a Pen object. The call to DrawImage draws the bitmap with its upper-left corner at (0, 0). The call to DrawEllipse uses the Pen object to draw a textured ellipse.

The following illustration shows the bitmap and the textured ellipse.



VB

```
Dim bitmap As New Bitmap("Texture1.jpg")
Dim tBrush As New TextureBrush(bitmap)
Dim texturedPen As New Pen(tBrush, 30)

e.Graphics.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height)
e.Graphics.DrawEllipse(texturedPen, 100, 20, 200, 100)
```

Compiling the Code

Create a Windows Form and handle the form's Paint event. Paste the preceding code into the Paint event handler. Replace `Texture.jpg` with an image valid on your system.

See Also

- Using a Pen to Draw Lines and Shapes
- Graphics and Drawing in Windows Forms

How to: Fill a Shape with a Solid Color

.NET Framework (current version)

To fill a shape with a solid color, create a SolidBrush object, and then pass that SolidBrush object as an argument to one of the fill methods of the Graphics class. The following example shows how to fill an ellipse with the color red.

Example

In the following code, the SolidBrush constructor takes a Color object as its only argument. The values used by the FromArgb method represent the alpha, red, green, and blue components of the color. Each of these values must be in the range 0 through 255. The first 255 indicates that the color is fully opaque, and the second 255 indicates that the red component is at full intensity. The two zeros indicate that the green and blue components both have an intensity of 0.

The four numbers (0, 0, 100, 60) passed to the FillEllipse method specify the location and size of the bounding rectangle for the ellipse. The rectangle has an upper-left corner of (0, 0), a width of 100, and a height of 60.

VB

```
Dim solidBrush As New SolidBrush( _
   Color.FromArgb(255, 255, 0, 0))
e.Graphics.FillEllipse(solidBrush, 0, 0, 100, 60)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Brush to Fill Shapes

How to: Fill a Shape with a Hatch Pattern

.NET Framework (current version)

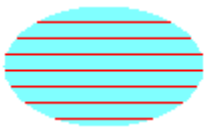
A hatch pattern is made from two colors: one for the background and one for the lines that form the pattern over the background. To fill a closed shape with a hatch pattern, use a HatchBrush object. The following example demonstrates how to fill an ellipse with a hatch pattern:

Example

The HatchBrush constructor takes three arguments: the hatch style, the color of the hatch line, and the color of the background. The hatch style argument can be any value from the HatchStyle enumeration. There are more than fifty elements in the HatchStyle enumeration; a few of those elements are shown in the following list:

- Horizontal
- Vertical
- ForwardDiagonal
- BackwardDiagonal
- Cross
- DiagonalCross

The following illustration shows the filled ellipse.



VB

```
Dim hBrush As New HatchBrush( _
   HatchStyle.Horizontal, _
   Color.Red, _
   Color.FromArgb(255, 128, 255, 255))
e.Graphics.FillEllipse(hBrush, 0, 0, 100, 60)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Brush to Fill Shapes

How to: Fill a Shape with an Image Texture

.NET Framework (current version)

You can fill a closed shape with a texture by using the Image class and the TextureBrush class.

Example

The following example fills an ellipse with an image. The code constructs an Image object, and then passes the address of that Image object as an argument to a TextureBrush constructor. The third statement scales the image, and the fourth statement fills the ellipse with repeated copies of the scaled image.

In the following code, the Transform property contains the transformation that is applied to the image before it is drawn. Assume that the original image has a width of 640 pixels and a height of 480 pixels. The transform shrinks the image to 75×75 by setting the horizontal and vertical scaling values.

Note

In the following example, the image size is 75×75, and the ellipse size is 150×250. Because the image is smaller than the ellipse it is filling, the ellipse is tiled with the image. Tiling means that the image is repeated horizontally and vertically until the boundary of the shape is reached. For more information about tiling, see How to: Tile a Shape with an Image.

VB

```
Dim image As New Bitmap("ImageFile.jpg")
Dim tBrush As New TextureBrush(image)
tBrush.Transform = New Matrix( _
   75.0F / 640.0F, _
   0.0F, _
   0.0F, _
   75.0F / 480.0F, _
   0.0F, _
   0.0F)
e.Graphics.FillEllipse(tBrush, New Rectangle(0, 150, 150, 250))
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Brush to Fill Shapes

How to: Tile a Shape with an Image

.NET Framework (current version)

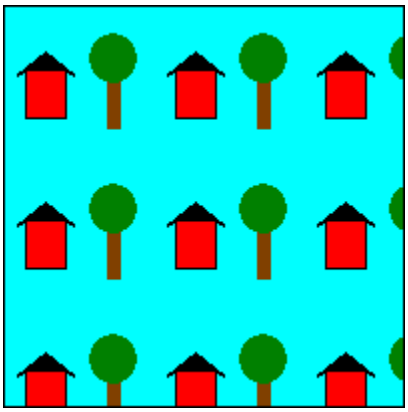
Just as tiles can be placed next to each other to cover a floor, rectangular images can be placed next to each other to fill (tile) a shape. To tile the interior of a shape, use a texture brush. When you construct a TextureBrush object, one of the arguments you pass to the constructor is an Image object. When you use the texture brush to paint the interior of a shape, the shape is filled with repeated copies of this image.

The wrap mode property of the TextureBrush object determines how the image is oriented as it is repeated in a rectangular grid. You can make all the tiles in the grid have the same orientation, or you can make the image flip from one grid position to the next. The flipping can be horizontal, vertical, or both. The following examples demonstrate tiling with different types of flipping.

To tile an image

- This example uses the following 75×75 image to tile a 200×200 rectangle.

- The following illustration shows how the rectangle is tiled with the image. Note that all tiles have the same orientation; there is no flipping.

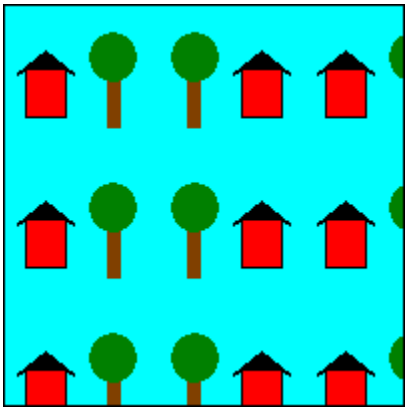


VB

```
Dim image As New Bitmap("HouseAndTree.gif")
Dim tBrush As New TextureBrush(image)
Dim blackPen As New Pen(Color.Black)
e.Graphics.FillRectangle(tBrush, New Rectangle(0, 0, 200, 200))
e.Graphics.DrawRectangle(blackPen, New Rectangle(0, 0, 200, 200))
```

To flip an image horizontally while tiling

- This example uses the same 75×75 image to fill a 200×200 rectangle. The wrap mode is set to flip the image horizontally. The following illustration shows how the rectangle is tiled with the image. Note that as you move from one tile to the next in a given row, the image is flipped horizontally.



VB

```
Dim image As New Bitmap("HouseAndTree.gif")
Dim tBrush As New TextureBrush(image)
Dim blackPen As New Pen(Color.Black)
tBrush.WrapMode = WrapMode.TileFlipX
e.Graphics.FillRectangle(tBrush, New Rectangle(0, 0, 200, 200))
e.Graphics.DrawRectangle(blackPen, New Rectangle(0, 0, 200, 200))
```

To flip an image vertically while tiling

- This example uses the same 75×75 image to fill a 200×200 rectangle. The wrap mode is set to flip the image vertically.

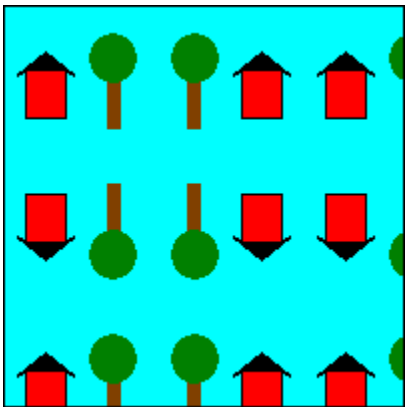
VB

```
Dim image As New Bitmap("HouseAndTree.gif")
Dim tBrush As New TextureBrush(image)
Dim blackPen As New Pen(Color.Black)
tBrush.WrapMode = WrapMode.TileFlipY
e.Graphics.FillRectangle(tBrush, New Rectangle(0, 0, 200, 200))
e.Graphics.DrawRectangle(blackPen, New Rectangle(0, 0, 200, 200))
```

To flip an image horizontally and vertically while tiling

- This example uses the same 75×75 image to tile a 200×200 rectangle. The wrap mode is set to flip the image both horizontally and vertically. The following illustration shows how the rectangle is tiled by the image. Note that as you move from one tile to the next in a given row, the image is flipped horizontally, and as you move from one tile to the

next in a given column, the image is flipped vertically.



VB

```
Dim image As New Bitmap("HouseAndTree.gif")
Dim tBrush As New TextureBrush(image)
Dim blackPen As New Pen(Color.Black)
tBrush.WrapMode = WrapMode.TileFlipXY
e.Graphics.FillRectangle(tBrush, New Rectangle(0, 0, 200, 200))
e.Graphics.DrawRectangle(blackPen, New Rectangle(0, 0, 200, 200))
```

See Also

Using a Brush to Fill Shapes

How to: Create a Linear Gradient

.NET Framework (current version)

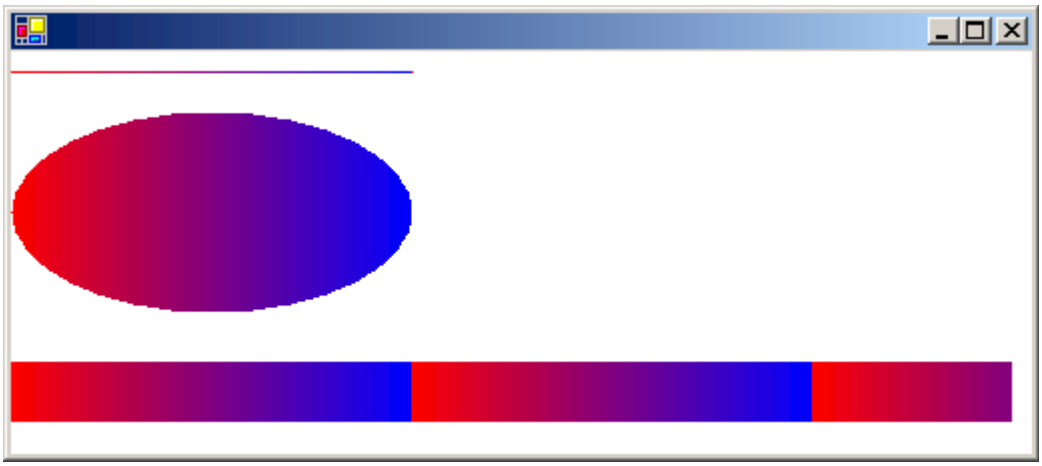
GDI+ provides horizontal, vertical, and diagonal linear gradients. By default, the color in a linear gradient changes uniformly. However, you can customize a linear gradient so that the color changes in a non-uniform fashion.

The following example fills a line, an ellipse, and a rectangle with a horizontal linear gradient brush.

The LinearGradientBrush constructor receives four arguments: two points and two colors. The first point (0, 10) is associated with the first color (red), and the second point (200, 10) is associated with the second color (blue). As you would expect, the line drawn from (0, 10) to (200, 10) changes gradually from red to blue.

The 10s in the points (50, 10) and (200, 10) are not important. What is important is that the two points have the same second coordinate — the line connecting them is horizontal. The ellipse and the rectangle also change gradually from red to blue as the horizontal coordinate goes from 0 to 200.

The following illustration shows the line, the ellipse, and the rectangle. Note that the color gradient repeats itself as the horizontal coordinate increases beyond 200.



To use horizontal linear gradients

- Pass in the opaque red and opaque blue as the third and fourth argument, respectively.

VB

```
Dim linGrBrush As New LinearGradientBrush( _
   New Point(0, 10), _
   New Point(200, 10), _
   Color.FromArgb(255, 255, 0, 0), _
   Color.FromArgb(255, 0, 0, 255))
Dim pen As New Pen(linGrBrush)

e.Graphics.DrawLine(pen, 0, 10, 200, 10)
e.Graphics.FillEllipse(linGrBrush, 0, 30, 200, 100)
e.Graphics.FillRectangle(linGrBrush, 0, 155, 500, 30)
```

In the preceding example, the color components change linearly as you move from a horizontal coordinate of 0 to a horizontal coordinate of 200. For example, a point whose first coordinate is halfway between 0 and 200 will have a blue component that is halfway between 0 and 255.

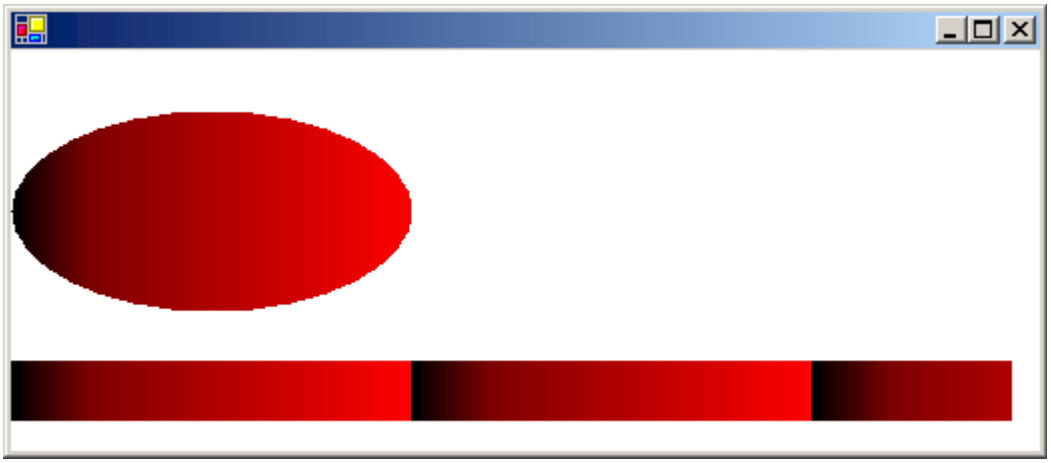
GDI+ allows you to adjust the way a color varies from one edge of a gradient to the other. Suppose you want to create a gradient brush that changes from black to red according to the following table.

Horizontal coordinate	RGB components
0	(0, 0, 0)
40	(128, 0, 0)
200	(255, 0, 0)

Note that the red component is at half intensity when the horizontal coordinate is only 20 percent of the way from 0 to 200.

The following example sets the Blend property of a LinearGradientBrush object to associate three relative intensities with three relative positions. As in the preceding table, a relative intensity of 0.5 is associated with a relative position of 0.2. The code fills an ellipse and a rectangle with the gradient brush.

The following illustration shows the resulting ellipse and rectangle.



To customize linear gradients

- Pass in the opaque black and opaque red as the third and fourth argument, respectively.

VB

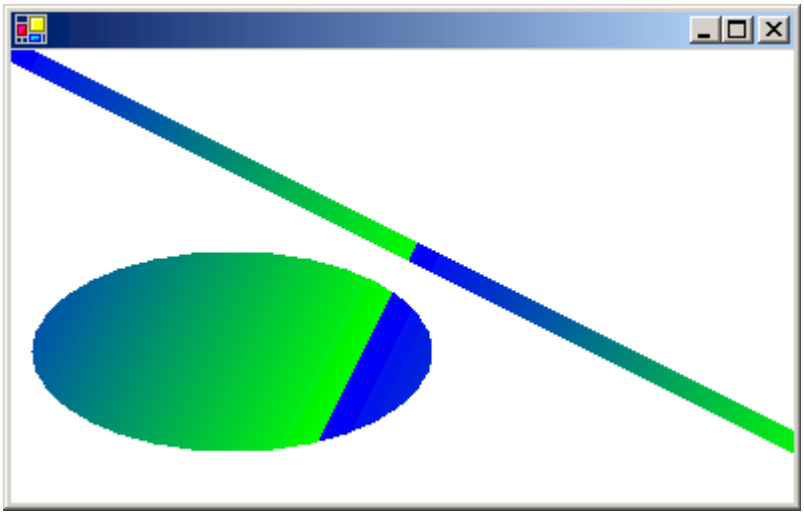
```
Dim linGrBrush As New LinearGradientBrush( _
   New Point(0, 10), _
   New Point(200, 10), _
   Color.FromArgb(255, 0, 0, 0), _
```

```
    Color.FromArgb(255, 255, 0, 0))

Dim relativeIntensities As Single() = {0.0F, 0.5F, 1.0F}
Dim relativePositions As Single() = {0.0F, 0.2F, 1.0F}

'Create a Blend object and assign it to linGrBrush.
Dim blend As New Blend()
blend.Factors = relativeIntensities
blend.Positions = relativePositions
linGrBrush.Blend = blend

e.Graphics.FillEllipse(linGrBrush, 0, 30, 200, 100)
e.Graphics.FillRectangle(linGrBrush, 0, 155, 500, 30)
```

The gradients in the preceding examples have been horizontal; that is, the color changes gradually as you move along any horizontal line. You can also define vertical gradients and diagonal gradients.

The following example passes the points (0, 0) and (200, 100) to a LinearGradientBrush constructor. The color blue is associated with (0, 0), and the color green is associated with (200, 100). A line (with pen width 10) and an ellipse are filled with the linear gradient brush.

The following illustration shows the line and the ellipse. Note that the color in the ellipse changes gradually as you move along any line that is parallel to the line passing through (0, 0) and (200, 100).



To create diagonal linear gradients

- Pass in the opaque blue and opaque green as the third and fourth argument, respectively.

VB

```
Dim linGrBrush As New LinearGradientBrush( _
   New Point(0, 0), _
   New Point(200, 100), _
   Color.FromArgb(255, 0, 0, 255), _
   Color.FromArgb(255, 0, 255, 0))
```

```
    ' opaque blue
    ' opaque green
    Dim pen As New Pen(linGrBrush, 10)

    e.Graphics.DrawLine(pen, 0, 0, 600, 300)
    e.Graphics.FillEllipse(linGrBrush, 10, 100, 200, 100)
```

See Also

Using a Gradient Brush to Fill Shapes
Graphics and Drawing in Windows Forms

How to: Create a Path Gradient

.NET Framework (current version)

The PathGradientBrush class allows you to customize the way you fill a shape with gradually changing colors. For example, you can specify one color for the center of a path and another color for the boundary of a path. You can also specify separate colors for each of several points along the boundary of a path.

Note

In GDI+, a path is a sequence of lines and curves maintained by a GraphicsPath object. For more information about GDI+ paths, see Graphics Paths in GDI+ and Constructing and Drawing Paths.

To fill an ellipse with a path gradient

- The following example fills an ellipse with a path gradient brush. The center color is set to blue and the boundary color is set to aqua. The following illustration shows the filled ellipse.

By default, a path gradient brush does not extend outside the boundary of the path. If you use the path gradient brush to fill a figure that extends beyond the boundary of the path, the area of the screen outside the path will not be filled.

The following illustration shows what happens if you change the FillEllipse call in the following code to `e.Graphics.FillRectangle(pthGrBrush, 0, 10, 200, 40)`.

VB

```
' Create a path that consists of a single ellipse.
Dim path As New GraphicsPath()
path.AddEllipse(0, 0, 140, 70)

' Use the path to construct a brush.
Dim pthGrBrush As New PathGradientBrush(path)

' Set the color at the center of the path to blue.
pthGrBrush.CenterColor = Color.FromArgb(255, 0, 0, 255)

' Set the color along the entire boundary
```

```
' of the path to aqua.
Dim colors As Color() = {Color.FromArgb(255, 0, 255, 255)}
pthGrBrush.SurroundColors = colors

e.Graphics.FillEllipse(pthGrBrush, 0, 0, 140, 70)
```

The preceding code example is designed for use with Windows Forms, and it requires the PaintEventArgs e, which is a parameter of PaintEventHandler.

To specify points on the boundary

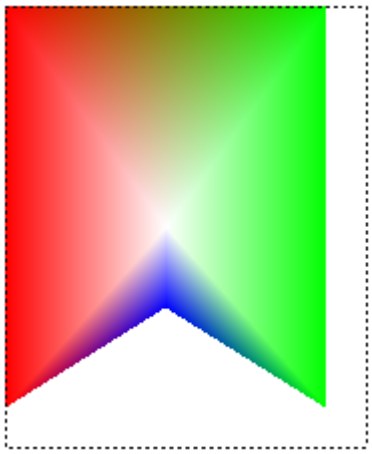
- The following example constructs a path gradient brush from a star-shaped path. The code sets the CenterColor property, which sets the color at the centroid of the star to red. Then the code sets the SurroundColors property to specify various colors (stored in the `colors` array) at the individual points in the `points` array. The final code statement fills the star-shaped path with the path gradient brush.

VB

```
' Put the points of a polygon in an array.
Dim points As Point() = { _
   New Point(75, 0), _
   New Point(100, 50), _
   New Point(150, 50), _
   New Point(112, 75), _
   New Point(150, 150), _
   New Point(75, 100), _
   New Point(0, 150), _
   New Point(37, 75), _
   New Point(0, 50), _
   New Point(50, 50)}

' Use the array of points to construct a path.
Dim path As New GraphicsPath()
path.AddLines(points)

' Use the path to construct a path gradient brush.
Dim pthGrBrush As New PathGradientBrush(path)

' Set the color at the center of the path to red.
pthGrBrush.CenterColor = Color.FromArgb(255, 255, 0, 0)

' Set the colors of the points in the array.
Dim colors As Color() = { _
   Color.FromArgb(255, 0, 0, 0), _
   Color.FromArgb(255, 0, 255, 0), _
   Color.FromArgb(255, 0, 0, 255), _
   Color.FromArgb(255, 255, 255, 255), _
   Color.FromArgb(255, 0, 0, 0), _
   Color.FromArgb(255, 0, 255, 0), _
   Color.FromArgb(255, 0, 0, 255), _
```

```
    Color.FromArgb(255, 255, 255, 255), _
    Color.FromArgb(255, 0, 0, 0), _
    Color.FromArgb(255, 0, 255, 0)}

pthGrBrush.SurroundColors = colors

' Fill the path with the path gradient brush.
e.Graphics.FillPath(pthGrBrush, path)
```

- The following example draws a path gradient without a GraphicsPath object in the code. The particular PathGradientBrush constructor in the example receives an array of points but does not require a GraphicsPath object. Also, note that the PathGradientBrush is used to fill a rectangle, not a path. The rectangle is larger than the closed path used to define the brush, so some of the rectangle is not painted by the brush. The following illustration shows the rectangle (dotted line) and the portion of the rectangle painted by the path gradient brush.



VB

```
' Construct a path gradient brush based on an array of points.
Dim ptsF As PointF() = { _
   New PointF(0, 0), _
   New PointF(160, 0), _
   New PointF(160, 200), _
   New PointF(80, 150), _
   New PointF(0, 200)}

Dim pBrush As New PathGradientBrush(ptsF)

' An array of five points was used to construct the path gradient
' brush. Set the color of each point in that array.
'Point (0, 0) is red
'Point (160, 0) is green
'Point (160, 200) is green
'Point (80, 150) is blue
'Point (0, 200) is red
Dim colors As Color() = { _
   Color.FromArgb(255, 255, 0, 0), _
   Color.FromArgb(255, 0, 255, 0), _
   Color.FromArgb(255, 0, 255, 0), _
   Color.FromArgb(255, 0, 0, 255), _
```

```
    Color.FromArgb(255, 255, 0, 0)}
pBrush.SurroundColors = colors

' Set the center color to white.
pBrush.CenterColor = Color.White

' Use the path gradient brush to fill a rectangle.
e.Graphics.FillRectangle(pBrush, New Rectangle(0, 0, 160, 200))
```

To customize a path gradient

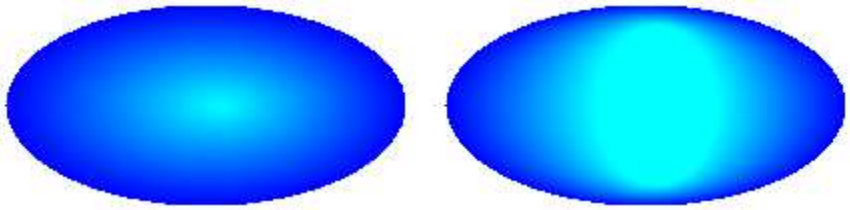
- One way to customize a path gradient brush is to set its FocusScales property. The focus scales specify an inner path that lies inside the main path. The center color is displayed everywhere inside that inner path rather than only at the center point.

  The following example creates a path gradient brush based on an elliptical path. The code sets the boundary color to blue, sets the center color to aqua, and then uses the path gradient brush to fill the elliptical path.

  Next, the code sets the focus scales of the path gradient brush. The x focus scale is set to 0.3, and the y focus scale is set to 0.8. The code calls the TranslateTransform method of a Graphics object so that the subsequent call to FillPath fills an ellipse that sits to the right of the first ellipse.

  To see the effect of the focus scales, imagine a small ellipse that shares its center with the main ellipse. The small (inner) ellipse is the main ellipse scaled (about its center) horizontally by a factor of 0.3 and vertically by a factor of 0.8. As you move from the boundary of the outer ellipse to the boundary of the inner ellipse, the color changes gradually from blue to aqua. As you move from the boundary of the inner ellipse to the shared center, the color remains aqua.

  The following illustration shows the output of the following code. The ellipse on the left is aqua only at the center point. The ellipse on the right is aqua everywhere inside the inner path.



VB

```
' Create a path that consists of a single ellipse.
Dim path As New GraphicsPath()
path.AddEllipse(0, 0, 200, 100)

' Create a path gradient brush based on the elliptical path.
Dim pthGrBrush As New PathGradientBrush(path)

' Set the color along the entire boundary to blue.
' Changed variable name from color
Dim blueColor As Color() = {Color.Blue}
pthGrBrush.SurroundColors = blueColor
```

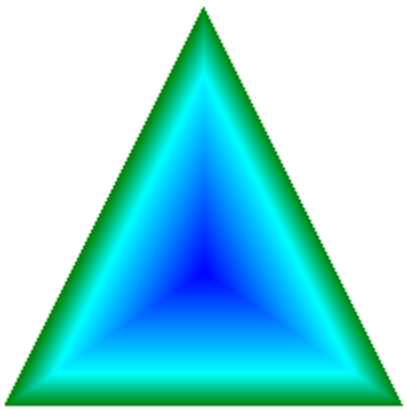
```
' Set the center color to aqua.
pthGrBrush.CenterColor = Color.Aqua

' Use the path gradient brush to fill the ellipse.
e.Graphics.FillPath(pthGrBrush, path)

' Set the focus scales for the path gradient brush.
pthGrBrush.FocusScales = New PointF(0.3F, 0.8F)

' Use the path gradient brush to fill the ellipse again.
' Show this filled ellipse to the right of the first filled ellipse.
e.Graphics.TranslateTransform(220.0F, 0.0F)
e.Graphics.FillPath(pthGrBrush, path)
```

To customize with interpolation

- Another way to customize a path gradient brush is to specify an array of interpolation colors and an array of interpolation positions.

  The following example creates a path gradient brush based on a triangle. The code sets the InterpolationColors property of the path gradient brush to specify an array of interpolation colors (dark green, aqua, blue) and an array of interpolation positions (0, 0.25, 1). As you move from the boundary of the triangle to the center point, the color changes gradually from dark green to aqua and then from aqua to blue. The change from dark green to aqua happens in 25 percent of the distance from dark green to blue.

  The following illustration shows the triangle filled with the custom path gradient brush.



VB

```
' Vertices of the outer triangle
Dim points As Point() = { _
   New Point(100, 0), _
   New Point(200, 200), _
   New Point(0, 200)}

' No GraphicsPath object is created. The PathGradientBrush
' object is constructed directly from the array of points.
Dim pthGrBrush As New PathGradientBrush(points)
```

```
' Create an array of colors containing dark green, aqua, and  blue.
Dim colors As Color() = { _
   Color.FromArgb(255, 0, 128, 0), _
   Color.FromArgb(255, 0, 255, 255), _
   Color.FromArgb(255, 0, 0, 255)}

' Dark green is at the boundary of the triangle.
' Aqua is 40 percent of the way from the boundary to the center point.
' Blue is at the center point.
Dim relativePositions As Single() = { _
   0.0F, _
   0.4F, _
   1.0F}

Dim colorBlend As New ColorBlend()
colorBlend.Colors = colors
colorBlend.Positions = relativePositions
pthGrBrush.InterpolationColors = colorBlend

' Fill a rectangle that is larger than the triangle
' specified in the Point array. The portion of the
' rectangle outside the triangle will not be painted.
e.Graphics.FillRectangle(pthGrBrush, 0, 0, 200, 200)
```

To set the center point

- By default, the center point of a path gradient brush is at the centroid of the path used to construct the brush. You can change the location of the center point by setting the CenterPoint property of the PathGradientBrush class.

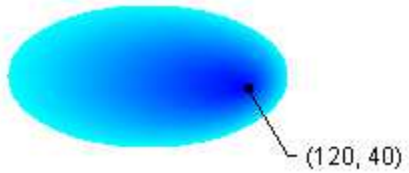
  The following example creates a path gradient brush based on an ellipse. The center of the ellipse is at (70, 35), but the center point of the path gradient brush is set to (120, 40).

VB

```
' Create a path that consists of a single ellipse.
Dim path As New GraphicsPath()
path.AddEllipse(0, 0, 140, 70)

' Use the path to construct a brush.
Dim pthGrBrush As New PathGradientBrush(path)

' Set the center point to a location that is not
' the centroid of the path.
pthGrBrush.CenterPoint = New PointF(120, 40)

' Set the color at the center of the path to blue.
pthGrBrush.CenterColor = Color.FromArgb(255, 0, 0, 255)

' Set the color along the entire boundary
' of the path to aqua.
```

```
Dim colors As Color() = {Color.FromArgb(255, 0, 255, 255)}
pthGrBrush.SurroundColors = colors

e.Graphics.FillEllipse(pthGrBrush, 0, 0, 140, 70)
```

The following illustration shows the filled ellipse and the center point of the path gradient brush.

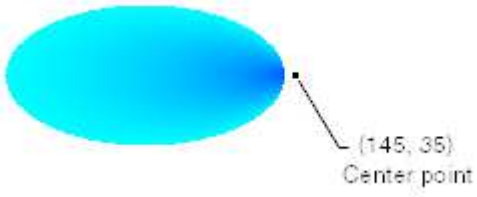


- You can set the center point of a path gradient brush to a location outside the path that was used to construct the brush. The following example replaces the call to set the CenterPoint property in the preceding code.

VB

```
pthGrBrush.CenterPoint = New PointF(145, 35)
```

The following illustration shows the output with this change.



In the preceding illustration, the points at the far right of the ellipse are not pure blue (although they are very close). The colors in the gradient are positioned as if the fill reached the point (145, 35) where the color would be pure blue (0, 0, 255). But the fill never reaches (145, 35) because a path gradient brush paints only inside its path.

Compiling the Code

The preceding examples are designed for use with Windows Forms, and they require PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Using a Gradient Brush to Fill Shapes

How to: Apply Gamma Correction to a Gradient

.NET Framework (current version)

You can enable gamma correction for a linear gradient brush by setting the brush's GammaCorrection property to **true**. You can disable gamma correction by setting the GammaCorrection property to **false**. Gamma correction is disabled by default.

Example

The example creates a linear gradient brush and uses that brush to fill two rectangles. The first rectangle is filled without gamma correction, and the second rectangle is filled with gamma correction.

The following illustration shows the two filled rectangles. The top rectangle, which does not have gamma correction, appears dark in the middle. The bottom rectangle, which has gamma correction, appears to have more uniform intensity.

VB

```
Dim linGrBrush As New LinearGradientBrush( _
   New Point(0, 10), _
   New Point(200, 10), _
   Color.Red, _
   Color.Blue)

e.Graphics.FillRectangle(linGrBrush, 0, 0, 200, 50)
linGrBrush.GammaCorrection = True
e.Graphics.FillRectangle(linGrBrush, 0, 60, 200, 50)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

- LinearGradientBrush
- Using a Gradient Brush to Fill Shapes

How to: Draw an Existing Bitmap to the Screen

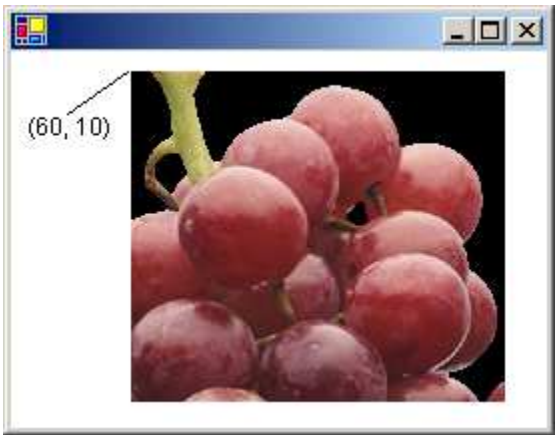
.NET Framework (current version)

You can easily draw an existing image on the screen. First you need to create a Bitmap object by using the bitmap constructor that takes a file name, Bitmap(String). This constructor accepts images with several different file formats, including BMP, GIF, JPEG, PNG, and TIFF. After you have created the Bitmap object, pass that Bitmap object to the DrawImage method of a Graphics object.

Example

This example creates a Bitmap object from a JPEG file and then draws the bitmap with its upper-left corner at (60, 10).

The following illustration shows the bitmap drawn at the specified location.



VB

```
Dim bitmap As New Bitmap("Grapes.jpg")
e.Graphics.DrawImage(bitmap, 60, 10)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Graphics and Drawing in Windows Forms
Working with Images, Bitmaps, Icons, and Metafiles

How to: Load and Display Metafiles

.NET Framework (current version)

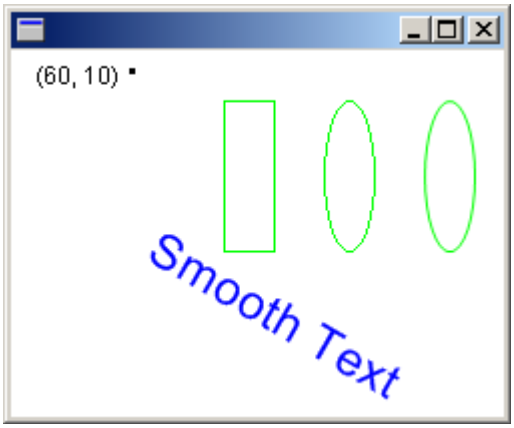
The Metafile class, which inherits from the Image class, provides methods for recording, displaying, and examining vector images.

Example

To display a vector image (metafile) on the screen, you need a Metafile object and a Graphics object. Pass the name of a file (or a stream) to a Metafile constructor. After you have created a Metafile object, pass that Metafile object to the DrawImage method of a Graphics object.

The example creates a Metafile object from an EMF (enhanced metafile) file and then draws the image with its upper-left corner at (60, 10).

The following illustration shows the metafile drawn at the specified location.



VB

```
Dim metafile As New Metafile("SampleMetafile.emf")
e.Graphics.DrawImage(metafile, 60, 10)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Working with Images, Bitmaps, Icons, and Metafiles

How to: Crop and Scale Images

.NET Framework (current version)

The Graphics class provides several DrawImage methods, some of which have source and destination rectangle parameters that you can use to crop and scale images.

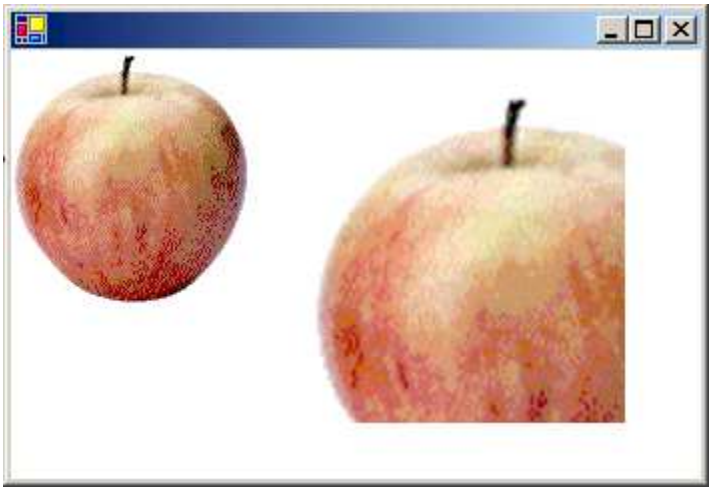
Example

The following example constructs an Image object from the disk file Apple.gif. The code draws the entire apple image in its original size. The code then calls the DrawImage method of a Graphics object to draw a portion of the apple image in a destination rectangle that is larger than the original apple image.

The DrawImage method determines which portion of the apple to draw by looking at the source rectangle, which is specified by the third, fourth, fifth, and sixth arguments. In this case, the apple is cropped to 75 percent of its width and 75 percent of its height.

The DrawImage method determines where to draw the cropped apple and how big to make the cropped apple by looking at the destination rectangle, which is specified by the second argument. In this case, the destination rectangle is 30 percent wider and 30 percent taller than the original image.

The following illustration shows the original apple and the scaled, cropped apple.



VB

```
Dim image As New Bitmap("Apple.gif")

' Draw the image unaltered with its upper-left corner at (0, 0).
e.Graphics.DrawImage(image, 0, 0)

' Make the destination rectangle 30 percent wider and
' 30 percent taller than the original image.
' Put the upper-left corner of the destination
' rectangle at (150, 20).
Dim width As Integer = image.Width
Dim height As Integer = image.Height
```

```
Dim destinationRect As New RectangleF( _
    150, _
    20, _
    1.3F * width, _
    1.3F * height)

' Draw a portion of the image. Scale that portion of the image
' so that it fills the destination rectangle.
Dim sourceRect As New RectangleF(0, 0, 0.75F * width, 0.75F * height)
e.Graphics.DrawImage( _
    image, _
    destinationRect, _
    sourceRect, _
    GraphicsUnit.Pixel)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Make sure to replace *Apple.gif* with an image file name and path that are valid on your system.

See Also

Images, Bitmaps, and Metafiles
Working with Images, Bitmaps, Icons, and Metafiles

How to: Rotate, Reflect, and Skew Images

.NET Framework (current version)

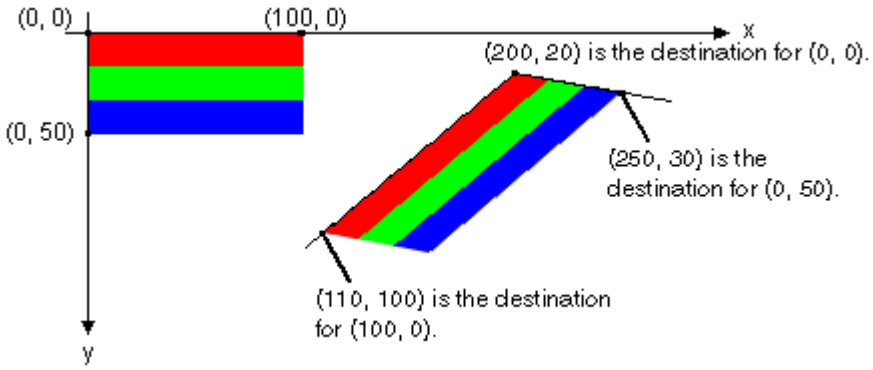
You can rotate, reflect, and skew an image by specifying destination points for the upper-left, upper-right, and lower-left corners of the original image. The three destination points determine an affine transformation that maps the original rectangular image to a parallelogram.

Example

For example, suppose the original image is a rectangle with upper-left corner at (0, 0), upper-right corner at (100, 0), and lower-left corner at (0, 50). Now suppose you map those three points to destination points as follows.

Original point	Destination point
Upper-left (0, 0)	(200, 20)
Upper-right (100, 0)	(110, 100)
Lower-left (0, 50)	(250, 30)

The following illustration shows the original image and the image mapped to the parallelogram. The original image has been skewed, reflected, rotated, and translated. The x-axis along the top edge of the original image is mapped to the line that runs through (200, 20) and (110, 100). The y-axis along the left edge of the original image is mapped to the line that runs through (200, 20) and (250, 30).



The following illustration shows a similar transformation applied to a photographic image.

The following illustration shows a similar transformation applied to a metafile.

The following example produces the images shown in the first illustration.

VB

```
' New Point(200, 20)  = destination for upper-left point of original
' New Point(110, 100) = destination for upper-right point of original
' New Point(250, 30)  = destination for lower-left point of original
Dim destinationPoints As Point() = { _
    New Point(200, 20), _
    New Point(110, 100), _
    New Point(250, 30)}

Dim image As New Bitmap("Stripes.bmp")

' Draw the image unaltered with its upper-left corner at (0, 0).
e.Graphics.DrawImage(image, 0, 0)

' Draw the image mapped to the parallelogram.
e.Graphics.DrawImage(image, destinationPoints)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Make sure to replace *Stripes.bmp* with the path to an image that is valid on your system.

See Also

Working with Images, Bitmaps, Icons, and Metafiles

How to: Use Interpolation Mode to Control Image Quality During Scaling

.NET Framework (current version)

The interpolation mode of a Graphics object influences the way GDI+ scales (stretches and shrinks) images. The InterpolationMode enumeration defines several interpolation modes, some of which are shown in the following list:

- NearestNeighbor
- Bilinear
- HighQualityBilinear
- Bicubic
- HighQualityBicubic

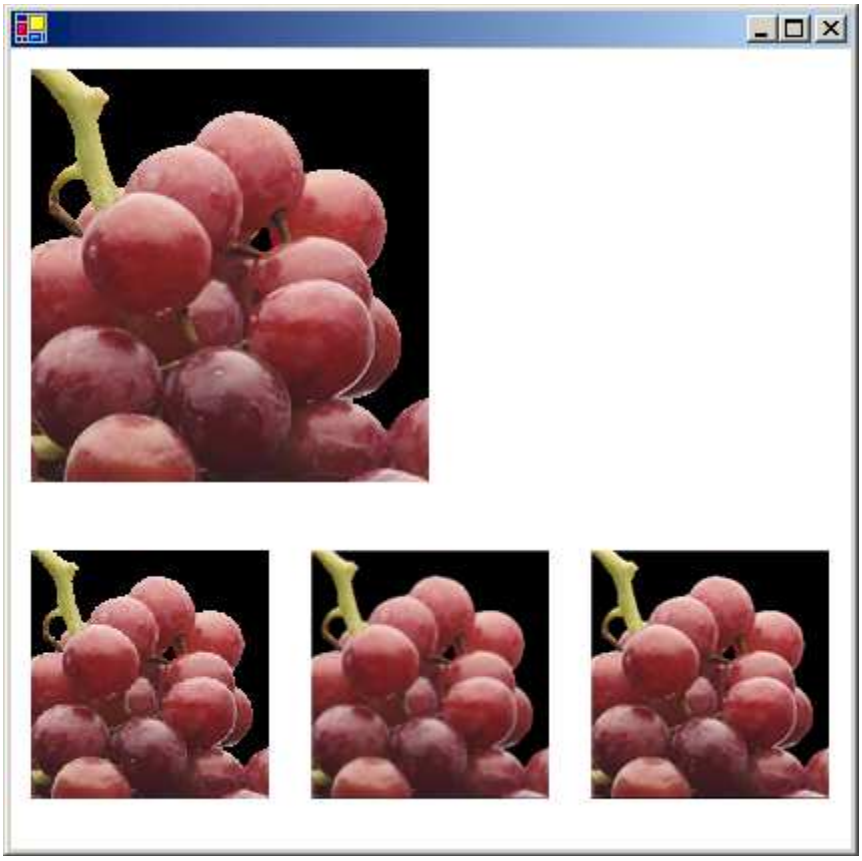
To stretch an image, each pixel in the original image must be mapped to a group of pixels in the larger image. To shrink an image, groups of pixels in the original image must be mapped to single pixels in the smaller image. The effectiveness of the algorithms that perform these mappings determines the quality of a scaled image. Algorithms that produce higher-quality scaled images tend to require more processing time. In the preceding list, NearestNeighbor is the lowest-quality mode and HighQualityBicubic is the highest-quality mode.

To set the interpolation mode, assign one of the members of the InterpolationMode enumeration to the InterpolationMode property of a Graphics object.

Example

The following example draws an image and then shrinks the image with three different interpolation modes.

The following illustration shows the original image and the three smaller images.



VB

```
Dim image As New Bitmap("GrapeBunch.bmp")
Dim width As Integer = image.Width
Dim height As Integer = image.Height

' Draw the image with no shrinking or stretching. Pass in the destination
' rectangle (2nd argument), the upper-left corner (3rd and 4th arguments),
' width (5th argument),  and height (6th argument) of the source
' rectangle.
e.Graphics.DrawImage( _
    image, _
    New Rectangle(10, 10, width, height), _
    0, _
    0, _
    width, _
    height, _
    GraphicsUnit.Pixel, _
    Nothing)

' Shrink the image using low-quality interpolation.
e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor

' Pass in the destination rectangle, and the upper-left corner, width,
' and height of the source rectangle as above.
e.Graphics.DrawImage( _
image, _
New Rectangle(10, 250, CInt(0.6 * width), CInt(0.6 * height)), _
0, _
```

```
    0, _
    width, _
    height, _
    GraphicsUnit.Pixel)

    ' Shrink the image using medium-quality interpolation.
    e.Graphics.InterpolationMode = InterpolationMode.HighQualityBilinear

    ' Pass in the destination rectangle, and the upper-left corner, width,
    ' and height of the source rectangle as above.
    e.Graphics.DrawImage( _
    image, _
    New Rectangle(150, 250, CInt(0.6 * width), _
    CInt(0.6 * height)), _
    0, _
    0, _
    width, _
    height, _
    GraphicsUnit.Pixel)

    ' Shrink the image using high-quality interpolation.
    e.Graphics.InterpolationMode = InterpolationMode.HighQualityBicubic

    ' Pass in the destination rectangle, and the upper-left corner, width,
    ' and height of the source rectangle as above.
    e.Graphics.DrawImage( _
        image, _
        New Rectangle(290, 250, CInt(0.6 * width), CInt(0.6 * height)), _
        0, _
        0, _
        width, _
        height, _
        GraphicsUnit.Pixel)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Images, Bitmaps, and Metafiles
Working with Images, Bitmaps, Icons, and Metafiles

How to: Create Thumbnail Images

.NET Framework (current version)

A thumbnail image is a small version of an image. You can create a thumbnail image by calling the GetThumbnailImage method of an Image object.

Example

The following example constructs an Image object from a JPG file. The original image has a width of 640 pixels and a height of 479 pixels. The code creates a thumbnail image that has a width of 100 pixels and a height of 100 pixels.

The following illustration shows the thumbnail image.

Note

In this example, a callback method is declared, but never used. This supports all versions of GDI+.

VB

```
Public Function ThumbnailCallback() As Boolean
                Return True
End Function

Private Sub GetThumbnail(ByVal e As PaintEventArgs)

                Dim callback As New Image.GetThumbnailImageAbort(AddressOf
ThumbnailCallback)
                Dim image As Image = New Bitmap("c:\FakePhoto.jpg")
                Dim pThumbnail As Image = image.GetThumbnailImage(100, 100, callback, New
IntPtr())
                e.Graphics.DrawImage(pThumbnail, 10, 10, pThumbnail.Width,
pThumbnail.Height)
End Sub
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. To run the example, follow these steps:

1. Create a new Windows Forms application.
2. Add the example code to the form.
3. Create a handler for the form's Paint event
4. In the Paint handler, call the `GetThumbnail` method and pass *e* for PaintEventArgs.
5. Find an image file that you want to make a thumbnail of.
6. In the `GetThumbnail` method, specify the path and file name to your image.
7. Press F5 to run the example.

   A 100 by 100 thumbnail image appears on the form.

See Also

Images, Bitmaps, and Metafiles
Working with Images, Bitmaps, Icons, and Metafiles

How to: Improve Performance by Avoiding Automatic Scaling

.NET Framework (current version)

GDI+ may automatically scale an image as you draw it, which would decrease performance. Alternatively, you can control the scaling of the image by passing the dimensions of the destination rectangle to the DrawImage method.

For example, the following call to the DrawImage method specifies an upper-left corner of (50, 30) but does not specify a destination rectangle.

VB

```
e.Graphics.DrawImage(image, 50, 30) ' upper-left corner at (50, 30)
```

Although this is the easiest version of the DrawImage method in terms of the number of required arguments, it is not necessarily the most efficient. If the resolution used by GDI+ (usually 96 dots per inch) is different from the resolution stored in the Image object, then the DrawImage method will scale the image. For example, suppose an Image object has a width of 216 pixels and a stored horizontal resolution value of 72 dots per inch. Because 216/72 is 3, DrawImage will scale the image so that it has a width of 3 inches at a resolution of 96 dots per inch. That is, DrawImage will display an image that has a width of 96x3 = 288 pixels.

Even if your screen resolution is different from 96 dots per inch, GDI+ will probably scale the image as if the screen resolution were 96 dots per inch. That is because a GDI+ Graphics object is associated with a device context, and when GDI+ queries the device context for the screen resolution, the result is usually 96, regardless of the actual screen resolution. You can avoid automatic scaling by specifying the destination rectangle in the DrawImage method.

Example

The following example draws the same image twice. In the first case, the width and height of the destination rectangle are not specified, and the image is automatically scaled. In the second case, the width and height (measured in pixels) of the destination rectangle are specified to be the same as the width and height of the original image. The following illustration shows the image rendered twice.

VB

```
Dim image As New Bitmap("Texture.jpg")

e.Graphics.DrawImage(image, 10, 10)
```

```
    e.Graphics.DrawImage(image, 120, 10, image.Width, image.Height)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Replace Texture.jpg with an image name and path that are valid on your system.

See Also

Images, Bitmaps, and Metafiles
Working with Images, Bitmaps, Icons, and Metafiles

How to: Read Image Metadata

.NET Framework (current version)

Some image files contain metadata that you can read to determine features of the image. For example, a digital photograph might contain metadata that you can read to determine the make and model of the camera used to capture the image. With GDI+, you can read existing metadata, and you can also write new metadata to image files.

GDI+ stores an individual piece of metadata in a PropertyItem object. You can read the PropertyItems property of an Image object to retrieve all the metadata from a file. The PropertyItems property returns an array of PropertyItem objects.

A PropertyItem object has the following four properties: **Id**, **Value**, **Len**, and **Type**.

Id

A tag that identifies the metadata item. Some values that can be assigned to Id are shown in the following table.

Hexadecimal value	Description
0x0320	Image title
0x010F	Equipment manufacturer
0x0110	Equipment model
0x9003	ExifDTOriginal
0x829A	Exif exposure time
0x5090	Luminance table
0x5091	Chrominance table

Value

An array of values. The format of the values is determined by the Type property.

Len

The length (in bytes) of the array of values pointed to by the Value property.

Type

The data type of the values in the array pointed to by the **Value** property. The formats indicated by the **Type** property values are shown in the following table

Numeric value	Description
1	A **Byte**
2	An array of **Byte** objects encoded as ASCII
3	A 16-bit integer
4	A 32-bit integer
5	An array of two **Byte** objects that represent a rational number
6	Not used
7	Undefined
8	Not used
9	**SLong**
10	**SRational**

Example

Description

The following code example reads and displays the seven pieces of metadata in the file *FakePhoto.jpg*. The second (index 1) property item in the list has Id 0x010F (equipment manufacturer) and Type 2 (ASCII-encoded byte array). The code example displays the value of that property item.

The code produces output similar to the following:

```
Property Item 0

id: 0x320

type: 2

length: 16 bytes
```

```
Property Item 1

id: 0x10f

type: 2

length: 17 bytes


Property Item 2

id: 0x110

type: 2

length: 7 bytes


Property Item 3

id: 0x9003

type: 2

length: 20 bytes


Property Item 4

id: 0x829a

type: 5

length: 8 bytes


Property Item 5

id: 0x5090

type: 3

length: 128 bytes


Property Item 6

id: 0x5091

type: 3

length: 128 bytes


The equipment make is Northwind Camera.
```

Code

VB

```
'Create an Image object.
Dim image As Bitmap = New Bitmap("c:\FakePhoto.jpg")

'Get the PropertyItems property from image.
Dim propItems As PropertyItem() = image.PropertyItems

'Set up the display.
Dim font As New Font("Arial", 12)
Dim blackBrush As New SolidBrush(Color.Black)
Dim X As Integer = 0
Dim Y As Integer = 0

'For each PropertyItem in the array, display the ID, type, and length.
Dim count As Integer = 0
Dim propItem As PropertyItem
For Each propItem In propItems
    e.Graphics.DrawString( _
       "Property Item " & count.ToString(), _
       font, _
       blackBrush, _
       X, Y)

    Y += font.Height

    e.Graphics.DrawString( _
       "   iD: 0x" & propItem.Id.ToString("x"), _
       font, _
       blackBrush, _
       X, Y)

    Y += font.Height

    e.Graphics.DrawString( _
       "   type: " & propItem.Type.ToString(), _
       font, _
       blackBrush, _
       X, Y)

    Y += font.Height

    e.Graphics.DrawString( _
       "   length: " & propItem.Len.ToString() & " bytes", _
       font, _
       blackBrush, _
       X, Y)

    Y += font.Height

    count += 1
```

```
Next propItem
'Convert the value of the second property to a string, and display it.
Dim encoding As New System.Text.ASCIIEncoding()
Dim manufacturer As String = encoding.GetString(propItems(1).Value)

e.Graphics.DrawString( _
   "The equipment make is " & manufacturer & ".", _
   font, _
   blackBrush, _
   X, Y)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Handle the form's Paint event and paste this code into the paint event handler. You must replace *FakePhoto.jpg* with an image name and path valid on your system and import the `System.Drawing.Imaging` namespace.

See Also

Images, Bitmaps, and Metafiles
Working with Images, Bitmaps, Icons, and Metafiles

How to: Create a Bitmap at Run Time

.NET Framework (current version)

This example creates and draws in a Bitmap object and displays it in an existing Windows Forms PictureBox control.

Example

VB

```
Private pictureBox1 As New PictureBox()

Public Sub CreateBitmapAtRuntime()
    pictureBox1.Size = New Size(210, 110)
    Me.Controls.Add(pictureBox1)


    Dim flag As New Bitmap(200, 100)
    Dim flagGraphics As Graphics = Graphics.FromImage(flag)
    Dim red As Integer = 0
    Dim white As Integer = 11
    While white <= 100
        flagGraphics.FillRectangle(Brushes.Red, 0, red, 200, 10)
        flagGraphics.FillRectangle(Brushes.White, 0, white, 200, 10)
        red += 20
        white += 20
    End While
    pictureBox1.Image = flag

End Sub
```

Compiling the Code

This example requires:

- A Windows Form that imports the System, System.Drawing and System.Windows.Forms assemblies.

See Also

Bitmap
Images, Bitmaps, and Metafiles

How to: Extract the Icon Associated with a File in Windows Forms

.NET Framework (current version)

Many files have embedded icons that provide a visual representation of the associated file type. For example, Microsoft Word documents contain an icon that identifies them as Word documents. When displaying files in a list control or table control, you may want to display the icon representing the file type next to each file name. You can do this easily by using the ExtractAssociatedIcon method.

Example

The following code example demonstrates how to extract the icon associated with a file and display the file name and its associated icon in a ListView control.

VB

```
Private listView1 As ListView
Private imageList1 As ImageList


Public Sub ExtractAssociatedIconEx()

    ' Initialize the ListView, ImageList and Form.
    listView1 = New ListView()
    imageList1 = New ImageList()
    listView1.Location = New Point(37, 12)
    listView1.Size = New Size(161, 242)
    listView1.SmallImageList = imageList1
    listView1.View = View.SmallIcon
    Me.ClientSize = New System.Drawing.Size(292, 266)
    Me.Controls.Add(Me.listView1)
    Me.Text = "Form1"

    ' Get the c:\ directory.
    Dim dir As New System.IO.DirectoryInfo("c:\")

    Dim item As ListViewItem
    listView1.BeginUpdate()
    Dim file As System.IO.FileInfo
    For Each file In dir.GetFiles()

        ' Set a default icon for the file.
        Dim iconForFile As Icon = SystemIcons.WinLogo

        item = New ListViewItem(file.Name, 1)

        ' Check to see if the image collection contains an image
```

```
            ' for this extension, using the extension as a key.
            If Not (imageList1.Images.ContainsKey(file.Extension)) Then

                ' If not, add the image to the image list.
                iconForFile = System.Drawing.Icon.ExtractAssociatedIcon(file.FullName)
                imageList1.Images.Add(file.Extension, iconForFile)
            End If
            item.ImageKey = file.Extension
            listView1.Items.Add(item)

        Next file
        listView1.EndUpdate()
    End Sub
```

Compiling the Code

To compile the example:

- Paste the preceding code into a Windows Form, and call the `ExtractAssociatedIconExample` method from the form's constructor or Load event-handling method.

  You will need to make sure that your form imports the System.IO namespace.

See Also

Images, Bitmaps, and Metafiles
ListView Control (Windows Forms)

How to: Draw Opaque and Semitransparent Lines

.NET Framework (current version)

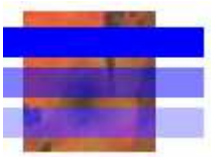
When you draw a line, you must pass a Pen object to the DrawLine method of the Graphics class. One of the parameters of the Pen constructor is a Color object. To draw an opaque line, set the alpha component of the color to 255. To draw a semitransparent line, set the alpha component to any value from 1 through 254.

When you draw a semitransparent line over a background, the color of the line is blended with the colors of the background. The alpha component specifies how the line and background colors are mixed; alpha values near 0 place more weight on the background colors, and alpha values near 255 place more weight on the line color.

Example

The following example draws a bitmap and then draws three lines that use the bitmap as a background. The first line uses an alpha component of 255, so it is opaque. The second and third lines use an alpha component of 128, so they are semitransparent; you can see the background image through the lines. The statement that sets the CompositingQuality property causes the blending for the third line to be done in conjunction with gamma correction.

The following illustration shows the output of the following code.



VB

```
Dim bitmap As New Bitmap("Texture1.jpg")
e.Graphics.DrawImage(bitmap, 10, 5, bitmap.Width, bitmap.Height)

Dim opaquePen As New Pen(Color.FromArgb(255, 0, 0, 255), 15)
Dim semiTransPen As New Pen(Color.FromArgb(128, 0, 0, 255), 15)

e.Graphics.DrawLine(opaquePen, 0, 20, 100, 20)
e.Graphics.DrawLine(semiTransPen, 0, 40, 100, 40)

e.Graphics.CompositingQuality = CompositingQuality.GammaCorrected
e.Graphics.DrawLine(semiTransPen, 0, 60, 100, 60)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Alpha Blending Lines and Fills
How to: Give Your Control a Transparent Background
How to: Draw with Opaque and Semitransparent Brushes

How to: Draw with Opaque and Semitransparent Brushes

.NET Framework (current version)

When you fill a shape, you must pass a Brush object to one of the fill methods of the Graphics class. The one parameter of the SolidBrush constructor is a Color object. To fill an opaque shape, set the alpha component of the color to 255. To fill a semitransparent shape, set the alpha component to any value from 1 through 254.

When you fill a semitransparent shape, the color of the shape is blended with the colors of the background. The alpha component specifies how the shape and background colors are mixed; alpha values near 0 place more weight on the background colors, and alpha values near 255 place more weight on the shape color.

Example

The following example draws a bitmap and then fills three ellipses that overlap the bitmap. The first ellipse uses an alpha component of 255, so it is opaque. The second and third ellipses use an alpha component of 128, so they are semitransparent; you can see the background image through the ellipses. The call that sets the CompositingQuality property causes the blending for the third ellipse to be done in conjunction with gamma correction.

The following illustration shows the output of the following code.

VB

```
Dim bitmap As New Bitmap("Texture1.jpg")
e.Graphics.DrawImage(bitmap, 50, 50, bitmap.Width, bitmap.Height)

Dim opaqueBrush As New SolidBrush(Color.FromArgb(255, 0, 0, 255))
Dim semiTransBrush As New SolidBrush(Color.FromArgb(128, 0, 0, 255))

e.Graphics.FillEllipse(opaqueBrush, 35, 45, 45, 30)
e.Graphics.FillEllipse(semiTransBrush, 86, 45, 45, 30)

e.Graphics.CompositingQuality = CompositingQuality.GammaCorrected
e.Graphics.FillEllipse(semiTransBrush, 40, 90, 86, 30)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

- Graphics and Drawing in Windows Forms
- Alpha Blending Lines and Fills
- How to: Give Your Control a Transparent Background
- How to: Draw Opaque and Semitransparent Lines

How to: Use Compositing Mode to Control Alpha Blending

.NET Framework (current version)

There may be times when you want to create an off-screen bitmap that has the following characteristics:

- Colors have alpha values that are less than 255.
- Colors are not alpha blended with each other as you create the bitmap.
- When you display the finished bitmap, colors in the bitmap are alpha blended with the background colors on the display device.

To create such a bitmap, construct a blank Bitmap object, and then construct a Graphics object based on that bitmap. Set the compositing mode of the Graphics object to CompositingMode.SourceCopy.

Example

The following example creates a Graphics object based on a Bitmap object. The code uses the Graphics object along with two semitransparent brushes (alpha = 160) to paint on the bitmap. The code fills a red ellipse and a green ellipse using the semitransparent brushes. The green ellipse overlaps the red ellipse, but the green is not blended with the red because the compositing mode of the Graphics object is set to SourceCopy.

The code draws the bitmap on the screen twice: once on a white background and once on a multicolored background. The pixels in the bitmap that are part of the two ellipses have an alpha component of 160, so the ellipses are blended with the background colors on the screen.

The following illustration shows the output of the code example. Note that the ellipses are blended with the background, but they are not blended with each other.

The code example contains this statement:

VB

```
bitmapGraphics.CompositingMode = CompositingMode.SourceCopy
```

If you want the ellipses to be blended with each other as well as with the background, change that statement to the following:

VB

```
bitmapGraphics.CompositingMode = CompositingMode.SourceOver
```

The following illustration shows the output of the revised code.

VB

```
' Create a blank bitmap.
Dim myBitmap As New Bitmap(180, 100)

' Create a Graphics object that we can use to draw on the bitmap.
Dim bitmapGraphics As Graphics = Graphics.FromImage(myBitmap)

' Create a red brush and a green brush, each with an alpha value of 160.
Dim redBrush As New SolidBrush(Color.FromArgb(160, 255, 0, 0))
Dim greenBrush As New SolidBrush(Color.FromArgb(160, 0, 255, 0))

' Set the compositing mode so that when we draw overlapping ellipses,
' the colors of the ellipses are not blended.
bitmapGraphics.CompositingMode = CompositingMode.SourceCopy

' Fill an ellipse using a red brush that has an alpha value of 160.
bitmapGraphics.FillEllipse(redBrush, 0, 0, 150, 70)

' Fill a second ellipse using a green brush that has an alpha value of
' 160. The green ellipse overlaps the red ellipse, but the green is not
' blended with the red.
bitmapGraphics.FillEllipse(greenBrush, 30, 30, 150, 70)

'Set the compositing quality of the form's Graphics object.
e.Graphics.CompositingQuality = CompositingQuality.GammaCorrected

' Draw a multicolored background.
Dim colorBrush As New SolidBrush(Color.Aqua)
e.Graphics.FillRectangle(colorBrush, 200, 0, 60, 100)
colorBrush.Color = Color.Yellow
e.Graphics.FillRectangle(colorBrush, 260, 0, 60, 100)
colorBrush.Color = Color.Fuchsia
e.Graphics.FillRectangle(colorBrush, 320, 0, 60, 100)

'Display the bitmap on a white background.
e.Graphics.DrawImage(myBitmap, 0, 0)

' Display the bitmap on a multicolored background.
e.Graphics.DrawImage(myBitmap, 200, 0)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

FromArgb
Alpha Blending Lines and Fills

How to: Use a Color Matrix to Set Alpha Values in Images

.NET Framework (current version)

The Bitmap class (which inherits from the Image class) and the ImageAttributes class provide functionality for getting and setting pixel values. You can use the ImageAttributes class to modify the alpha values for an entire image, or you can call the SetPixel method of the Bitmap class to modify individual pixel values.

Example

The ImageAttributes class has many properties that you can use to modify images during rendering. In the following example, an ImageAttributes object is used to set all the alpha values to 80 percent of what they were. This is done by initializing a color matrix and setting the alpha scaling value in the matrix to 0.8. The address of the color matrix is passed to the SetColorMatrix method of the ImageAttributes object, and the ImageAttributes object is passed to the DrawString method of the Graphics object.

During rendering, the alpha values in the bitmap are converted to 80 percent of what they were. This results in an image that is blended with the background. As the following illustration shows, the bitmap image looks transparent; you can see the solid black line through it.

Where the image is over the white portion of the background, the image has been blended with the color white. Where the image crosses the black line, the image is blended with the color black.

VB

```
' Create the Bitmap object and load it with the texture image.
Dim bitmap As New Bitmap("Texture.jpg")

' Initialize the color matrix.
' Note the value 0.8 in row 4, column 4.
Dim matrixItems As Single()() = { _
   New Single() {1, 0, 0, 0, 0}, _
   New Single() {0, 1, 0, 0, 0}, _
   New Single() {0, 0, 1, 0, 0}, _
   New Single() {0, 0, 0, 0.8F, 0}, _
   New Single() {0, 0, 0, 0, 1}}

Dim colorMatrix As New ColorMatrix(matrixItems)

' Create an ImageAttributes object and set its color matrix.
Dim imageAtt As New ImageAttributes()
imageAtt.SetColorMatrix( _
```

```
      colorMatrix, _
      ColorMatrixFlag.Default, _
      ColorAdjustType.Bitmap)

   ' First draw a wide black line.
   e.Graphics.DrawLine( _
      New Pen(Color.Black, 25), _
      New Point(10, 35), _
      New Point(200, 35))

   ' Now draw the semitransparent bitmap image.
   Dim iWidth As Integer = bitmap.Width
   Dim iHeight As Integer = bitmap.Height

   ' Pass in the destination rectangle (2nd argument) and the x _
   ' coordinate (3rd argument), x coordinate (4th argument), width _
   ' (5th argument), and height (6th argument) of the source rectangle.
   e.Graphics.DrawImage( _
      bitmap, _
      New Rectangle(30, 0, iWidth, iHeight), _
      0.0F, _
      0.0F, _
      iWidth, _
      iHeight, _
      GraphicsUnit.Pixel, _
      imageAtt)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Graphics and Drawing in Windows Forms
Alpha Blending Lines and Fills

How to: Construct Font Families and Fonts

.NET Framework (current version)

GDI+ groups fonts with the same typeface but different styles into font families. For example, the Arial font family contains the following fonts:

- Arial Regular
- Arial Bold
- Arial Italic
- Arial Bold Italic

GDI+ uses four styles to form families: regular, bold, italic, and bold italic. Adjectives such as *narrow* and *rounded* are not considered styles; rather they are part of the family name. For example, Arial Narrow is a font family with the following members:

- Arial Narrow Regular
- Arial Narrow Bold
- Arial Narrow Italic
- Arial Narrow Bold Italic

Before you can draw text with GDI+, you need to construct a FontFamily object and a Font object. The FontFamily object specifies the typeface (for example, Arial), and the Font object specifies the size, style, and units.

Example

The following example constructs a regular style Arial font with a size of 16 pixels. In the following code, the first argument passed to the Font constructor is the FontFamily object. The second argument specifies the size of the font measured in units identified by the fourth argument. The third argument identifies the style.

Pixel is a member of the GraphicsUnit enumeration, and Regular is a member of the FontStyle enumeration.

VB

```
Dim fontFamily As New FontFamily("Arial")
Dim font As New Font( _
   fontFamily, _
   16, _
   FontStyle.Regular, _
   GraphicsUnit.Pixel)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Using Fonts and Text
Graphics and Drawing in Windows Forms

How to: Draw Text at a Specified Location

.NET Framework (current version)

When you perform custom drawing, you can draw text in a single horizontal line starting at a specified point. You can draw text in this manner by using the DrawString overloaded method of the Graphics class that takes a Point or PointF parameter. The DrawString method also requires a Brush and Font

You can also use the DrawText overloaded method of the TextRenderer that takes a Point. DrawText also requires a Color and a Font.

The following illustration shows the output of text drawn at a specified point when you use the DrawString overloaded method.

To draw a line of text with GDI+

1. Use the DrawString method, passing the text you want, Point or PointF, Font, and Brush.

VB

```
Dim font1 As New Font("Times New Roman", 24, FontStyle.Bold, GraphicsUnit.Pixel)
Try
    Dim pointF1 As New PointF(30, 10)
    e.Graphics.DrawString("Hello", font1, Brushes.Blue, pointF1)
Finally
    font1.Dispose()
End Try
```

To draw a line of text with GDI

1. Use the DrawText method, passing the text you want, Point, Font, and Color.

VB

```
Dim font As New Font("Times New Roman", 24, FontStyle.Bold, GraphicsUnit.Pixel)
Try
    Dim point1 As New Point(30, 10)
    TextRenderer.DrawText(e.Graphics, "Hello", font, point1, Color.Blue)
Finally
    font.Dispose()
End Try
```

Compiling the Code

The previous examples require:

- PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

How to: Draw Text with GDI
Using Fonts and Text
How to: Construct Font Families and Fonts
How to: Draw Wrapped Text in a Rectangle

How to: Draw Wrapped Text in a Rectangle

.NET Framework (current version)

You can draw wrapped text in a rectangle by using the DrawString overloaded method of the Graphics class that takes a Rectangle or RectangleF parameter. You will also use a Brush and a Font.

You can also draw wrapped text in a rectangle by using the DrawText overloaded method of the TextRenderer that takes a Rectangle and a TextFormatFlags parameter. You will also use a Color and a Font.

The following illustration shows the output of text drawn in the rectangle when you use the DrawString method.

To draw wrapped text in a rectangle with GDI+

1. Use the DrawString overloaded method, passing the text you want, Rectangle or RectangleF, Font and Brush.

VB

```
Dim text1 As String = "Draw text in a rectangle by passing a RectF to the DrawString
method."
Dim font1 As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
Try
    Dim rectF1 As New RectangleF(30, 10, 100, 122)
    e.Graphics.DrawString(text1, font1, Brushes.Blue, rectF1)
    e.Graphics.DrawRectangle(Pens.Black, Rectangle.Round(rectF1))
Finally
    font1.Dispose()
End Try
```

To draw wrapped text in a rectangle with GDI

1. Use the TextFormatFlags enumeration value to specify the text should be wrapped with the DrawText overloaded method, passing the text you want, Rectangle, Font and Color.

VB

```
Dim text2 As String = _
```

```
    "Draw text in a rectangle by passing a RectF to the DrawString method."
Dim font2 As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
Try
    Dim rect2 As New Rectangle(30, 10, 100, 122)

    ' Specify the text is wrapped.
    Dim flags As TextFormatFlags = TextFormatFlags.WordBreak
    TextRenderer.DrawText(e.Graphics, text2, font2, rect2, Color.Blue, flags)
    e.Graphics.DrawRectangle(Pens.Black, Rectangle.Round(rect2))
Finally
    font2.Dispose()
End Try
```

Compiling the Code

The previous examples require:

- PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

How to: Draw Text with GDI
Using Fonts and Text
How to: Construct Font Families and Fonts
How to: Draw Text at a Specified Location

How to: Draw Text with GDI

.NET Framework (current version)

With the DrawText method in the TextRenderer class, you can access GDI functionality for drawing text on a form or control. GDI text rendering typically offers better performance and more accurate text measuring than GDI+.

Note

The DrawText methods of the TextRenderer class are not supported for printing. When printing, always use the DrawString methods of the Graphics class.

Example

The following code example demonstrates how to draw text on multiple lines within a rectangle using the DrawText method.

VB

```
Private Sub RenderText6(ByVal e As PaintEventArgs)
    Dim flags As TextFormatFlags = TextFormatFlags.Bottom Or _
        TextFormatFlags.EndEllipsis
    TextRenderer.DrawText(e.Graphics, _
    "This is some text that will be clipped at the end.", _
    Me.Font, New Rectangle(10, 10, 100, 50), SystemColors.ControlText, flags)

End Sub
```

To render text with the TextRenderer class, you need an IDeviceContext, such as a Graphics and a Font, a location to draw the text, and the color in which it should be drawn. Optionally, you can specify the text formatting by using the TextFormatFlags enumeration.

For more information about obtaining a Graphics, see How to: Create Graphics Objects for Drawing. For more information about constructing a Font, see How to: Construct Font Families and Fonts.

Compiling the Code

The preceding code example is designed for use with Windows Forms, and it requires the PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

TextRenderer
Font

Color
Color
Using Fonts and Text

How to: Align Drawn Text

.NET Framework (current version)

When you perform custom drawing, you may often want to center drawn text on a form or control. You can easily align text drawn with the DrawString or DrawText methods by creating the correct formatting object and setting the appropriate format flags.

To draw centered text with GDI+ (DrawString)

1. Use a StringFormat with the appropriate DrawString method to specify centered text.

VB

```
Dim text1 As String = "Use StringFormat and Rectangle objects to" & _
    " center text in a rectangle."
Dim font1 As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
Try
    Dim rect1 As New Rectangle(10, 10, 130, 140)

    ' Create a StringFormat object with the each line of text, and the block
    ' of text centered on the page.
    Dim stringFormat As New StringFormat()
    stringFormat.Alignment = StringAlignment.Center
    stringFormat.LineAlignment = StringAlignment.Center

    ' Draw the text and the surrounding rectangle.
    e.Graphics.DrawString(text1, font1, Brushes.Blue, rect1, stringFormat)
    e.Graphics.DrawRectangle(Pens.Black, rect1)
Finally
    font1.Dispose()
End Try
```

To draw centered text with GDI (DrawText)

1. Use the TextFormatFlags enumeration for wrapping as well as vertically and horizontally centering text with the appropriate DrawText method.

VB

```
Dim text2 As String = "Use TextFormatFlags and Rectangle objects to" & _
        " center text in a rectangle."

Dim font2 As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
Try
    Dim rect2 As New Rectangle(150, 10, 130, 140)
```

```
        ' Create a TextFormatFlags with word wrapping, horizontal center and
        ' vertical center specified.
        Dim flags As TextFormatFlags = TextFormatFlags.HorizontalCenter Or _
            TextFormatFlags.VerticalCenter Or TextFormatFlags.WordBreak

        ' Draw the text and the surrounding rectangle.
        TextRenderer.DrawText(e.Graphics, text2, font2, rect2, Color.Blue, flags)
        e.Graphics.DrawRectangle(Pens.Black, rect2)
    Finally
        font2.Dispose()
    End Try
```

Compiling the Code

The preceding code examples are designed for use with Windows Forms, and they require PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

How to: Draw Text with GDI
Using Fonts and Text
How to: Construct Font Families and Fonts

How to: Create Vertical Text

.NET Framework (current version)

You can use a StringFormat object to specify that text be drawn vertically rather than horizontally.

Example

The following example assigns the value DirectionVertical to the FormatFlags property of a StringFormat object. That StringFormat object is passed to the DrawString method of the Graphics class. The value DirectionVertical is a member of the StringFormatFlags enumeration.

The following illustration shows the vertical text.

VB

```
Dim myText As String = "Vertical text"

Dim fontFamily As New FontFamily("Lucida Console")
Dim font As New Font( _
   fontFamily, _
   14, _
   FontStyle.Regular, _
   GraphicsUnit.Point)
Dim pointF As New PointF(40, 10)
Dim stringFormat As New StringFormat()
Dim solidBrush As New SolidBrush(Color.FromArgb(255, 0, 0, 255))

stringFormat.FormatFlags = StringFormatFlags.DirectionVertical

e.Graphics.DrawString(myText, font, solidBrush, pointF, stringFormat)
```

Compiling the Code

- The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e* , which is a

parameter of PaintEventHandler.

See Also

How to: Draw Text with GDI

How to: Set Tab Stops in Drawn Text

.NET Framework (current version)

You can set tab stops for text by calling the SetTabStops method of a StringFormat object and then passing that StringFormat object to the DrawString method of the Graphics class.

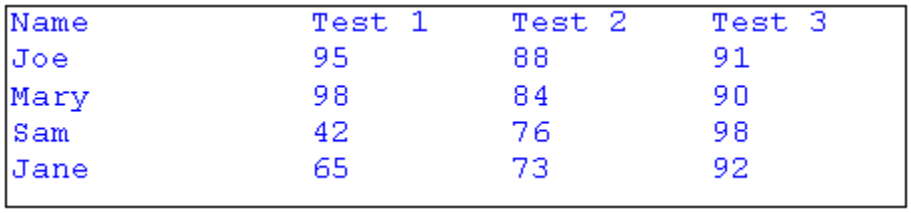
Note

The System.Windows.Forms.TextRenderer does not support adding tab stops to drawn text, although you can expand existing tab stops using the TextFormatFlags.ExpandTabs flag.

Example

The following example sets tab stops at 150, 250, and 350. Then, the code displays a tabbed list of names and test scores.

The following illustration shows the tabbed text.



The following code passes two arguments to the SetTabStops method. The second argument is an array that contains tab offsets. The first argument passed to SetTabStops is 0, which indicates that the first offset in the array is measured from position 0, the left edge of the bounding rectangle.

VB

```
Dim myText As String = _
   "Name" & ControlChars.Tab & _
   "Test 1" & ControlChars.Tab & _
   "Test 2" & ControlChars.Tab & _
   "Test 3" & ControlChars.Cr

myText = myText & "Joe" & ControlChars.Tab & _
                  "95" & ControlChars.Tab & _
                  "88" & ControlChars.Tab & _
                  "91" & ControlChars.Cr
myText = myText & "Mary" & ControlChars.Tab & _
                  "98" & ControlChars.Tab & _
                  "84" & ControlChars.Tab & _
                  "90" & ControlChars.Cr
myText = myText & "Sam" & ControlChars.Tab & _
                  "42" & ControlChars.Tab & _
                  "76" & ControlChars.Tab & _
```

```
                    "98" & ControlChars.Cr
myText = myText & "Jane" & ControlChars.Tab & _
                    "65" & ControlChars.Tab & _
                    "73" & ControlChars.Tab & _
                    "92" & ControlChars.Cr

Dim fontFamily As New FontFamily("Courier New")
Dim font As New Font( _
   fontFamily, _
   12, _
   FontStyle.Regular, _
   GraphicsUnit.Point)
Dim rect As New Rectangle(10, 10, 450, 100)
Dim stringFormat As New StringFormat()
Dim solidBrush As New SolidBrush(Color.FromArgb(255, 0, 0, 255))
Dim tabs As Single() = {150, 100, 100, 100}

stringFormat.SetTabStops(0, tabs)

e.Graphics.DrawString(myText, font, solidBrush, RectangleF.op_implicit(rect),
stringFormat)

Dim pen As Pen = Pens.Black
e.Graphics.DrawRectangle(pen, rect)
```

Compiling the Code

- The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Using Fonts and Text
How to: Draw Text with GDI

How to: Enumerate Installed Fonts

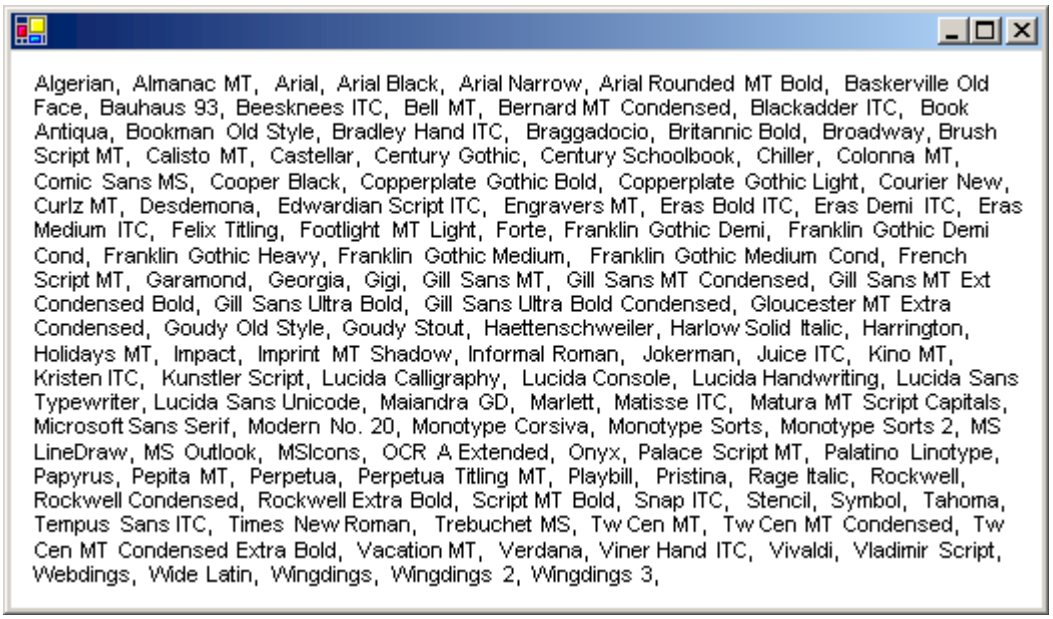
.NET Framework (current version)

The InstalledFontCollection class inherits from the FontCollection abstract base class. You can use an InstalledFontCollection object to enumerate the fonts installed on the computer. The Families property of an InstalledFontCollection object is an array of FontFamily objects.

Example

The following example lists the names of all the font families installed on the computer. The code retrieves the Name property of each FontFamily object in the array returned by the Families property. As the family names are retrieved, they are concatenated to form a comma-separated list. Then the DrawString method of the Graphics class draws the comma-separated list in a rectangle.

If you run the example code, the output will be similar to that shown in the following illustration.



VB

```
Dim fontFamily As New FontFamily("Arial")
Dim font As New Font( _
   fontFamily, _
   8, _
   FontStyle.Regular, _
   GraphicsUnit.Point)
Dim rectF As New RectangleF(10, 10, 500, 500)
Dim solidBrush As New SolidBrush(Color.Black)

Dim familyName As String
Dim familyList As String = ""
Dim fontFamilies() As FontFamily
```

```
Dim installedFontCollection As New InstalledFontCollection()

' Get the array of FontFamily objects.
fontFamilies = installedFontCollection.Families

' The loop below creates a large string that is a comma-separated
' list of all font family names.
Dim count As Integer = fontFamilies.Length
Dim j As Integer

While j < count
    familyName = fontFamilies(j).Name
    familyList = familyList & familyName
    familyList = familyList & ",  "
    j += 1
End While

' Draw the large string (list of all families) in a rectangle.
e.Graphics.DrawString(familyList, font, solidBrush, rectF)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler. In addition, you should import the System.Drawing.Text namespace.

See Also

Using Fonts and Text

How to: Create a Private Font Collection

.NET Framework (current version)

The PrivateFontCollection class inherits from the FontCollection abstract base class. You can use a PrivateFontCollection object to maintain a set of fonts specifically for your application. A private font collection can include installed system fonts as well as fonts that have not been installed on the computer. To add a font file to a private font collection, call the AddFontFile method of a PrivateFontCollection object.

The Families property of a PrivateFontCollection object contains an array of FontFamily objects.

The number of font families in a private font collection is not necessarily the same as the number of font files that have been added to the collection. For example, suppose you add the files ArialBd.tff, Times.tff, and TimesBd.tff to a collection. There will be three files but only two families in the collection because Times.tff and TimesBd.tff belong to the same family.

Example

The following example adds the following three font files to a PrivateFontCollection object:

- C:*systemroot*\Fonts\Arial.tff (Arial, regular)
- C:*systemroot*\Fonts\CourBI.tff (Courier New, bold italic)
- C:*systemroot*\Fonts\TimesBd.tff (Times New Roman, bold)

The code retrieves an array of FontFamily objects from the Families property of the PrivateFontCollection object.

For each FontFamily object in the collection, the code calls the IsStyleAvailable method to determine whether various styles (regular, bold, italic, bold italic, underline, and strikeout) are available. The arguments passed to the IsStyleAvailable method are members of the FontStyle enumeration.

If a given family/style combination is available, a Font object is constructed using that family and style. The first argument passed to the Font constructor is the font family name (not a FontFamily object as is the case for other variations of the Font constructor). After the Font object is constructed, it is passed to the DrawString method of the Graphics class to display the family name along with the name of the style.

The output of the following code is similar to the output shown in the following illustration.

Arial.tff (which was added to the private font collection in the following code example) is the font file for the Arial regular style. Note, however, that the program output shows several available styles other than regular for the Arial font family. That is because GDI+ can simulate the bold, italic, and bold italic styles from the regular style. GDI+ can also produce underlines and strikeouts from the regular style.

Similarly, GDI+ can simulate the bold italic style from either the bold style or the italic style. The program output shows that the bold italic style is available for the Times family even though TimesBd.tff (Times New Roman, bold) is the only Times file in the collection.

VB

```
Dim pointF As New PointF(10, 0)
Dim solidBrush As New SolidBrush(Color.Black)

Dim count As Integer = 0
Dim familyName As String = ""
Dim familyNameAndStyle As String
Dim fontFamilies() As FontFamily
Dim privateFontCollection As New PrivateFontCollection()

' Add three font files to the private collection.
privateFontCollection.AddFontFile("D:\systemroot\Fonts\Arial.ttf")
privateFontCollection.AddFontFile("D:\systemroot\Fonts\CourBI.ttf")
privateFontCollection.AddFontFile("D:\systemroot\Fonts\TimesBD.ttf")

' Get the array of FontFamily objects.
fontFamilies = privateFontCollection.Families

' How many objects in the fontFamilies array?
count = fontFamilies.Length

' Display the name of each font family in the private collection
' along with the available styles for that font family.
Dim j As Integer

While j < count
    ' Get the font family name.
    familyName = fontFamilies(j).Name

    ' Is the regular style available?
```

```
        If fontFamilies(j).IsStyleAvailable(FontStyle.Regular) Then
            familyNameAndStyle = ""
            familyNameAndStyle = familyNameAndStyle & familyName
            familyNameAndStyle = familyNameAndStyle & " Regular"

            Dim regFont As New Font( _
               familyName, _
               16, _
               FontStyle.Regular, _
               GraphicsUnit.Pixel)

            e.Graphics.DrawString( _
               familyNameAndStyle, _
               regFont, _
               solidBrush, _
               pointF)

            pointF.Y += regFont.Height
        End If

        ' Is the bold style available?
        If fontFamilies(j).IsStyleAvailable(FontStyle.Bold) Then
            familyNameAndStyle = ""
            familyNameAndStyle = familyNameAndStyle & familyName
            familyNameAndStyle = familyNameAndStyle & " Bold"

            Dim boldFont As New Font( _
               familyName, _
               16, _
               FontStyle.Bold, _
               GraphicsUnit.Pixel)

            e.Graphics.DrawString( _
               familyNameAndStyle, _
               boldFont, _
               solidBrush, _
               pointF)

            pointF.Y += boldFont.Height
        End If

        ' Is the italic style available?
        If fontFamilies(j).IsStyleAvailable(FontStyle.Italic) Then
            familyNameAndStyle = ""
            familyNameAndStyle = familyNameAndStyle & familyName
            familyNameAndStyle = familyNameAndStyle & " Italic"

            Dim italicFont As New Font( _
               familyName, _
               16, _
               FontStyle.Italic, _
               GraphicsUnit.Pixel)

            e.Graphics.DrawString( _
```

```
            familyNameAndStyle, _
            italicFont, _
            solidBrush, pointF)

        pointF.Y += italicFont.Height
    End If

    ' Is the bold italic style available?
    If fontFamilies(j).IsStyleAvailable(FontStyle.Italic) And _
       fontFamilies(j).IsStyleAvailable(FontStyle.Bold) Then
        familyNameAndStyle = ""
        familyNameAndStyle = familyNameAndStyle & familyName
        familyNameAndStyle = familyNameAndStyle & "BoldItalic"

        Dim italicFont As New Font( _
            familyName, _
            16, _
            FontStyle.Italic Or FontStyle.Bold, _
            GraphicsUnit.Pixel)

        e.Graphics.DrawString( _
            familyNameAndStyle, _
            italicFont, _
            solidBrush, _
            pointF)

        pointF.Y += italicFont.Height
    End If
    ' Is the underline style available?
    If fontFamilies(j).IsStyleAvailable(FontStyle.Underline) Then
        familyNameAndStyle = ""
        familyNameAndStyle = familyNameAndStyle & familyName
        familyNameAndStyle = familyNameAndStyle & " Underline"

        Dim underlineFont As New Font( _
            familyName, _
            16, _
            FontStyle.Underline, _
            GraphicsUnit.Pixel)

        e.Graphics.DrawString( _
            familyNameAndStyle, _
            underlineFont, _
            solidBrush, _
            pointF)

        pointF.Y += underlineFont.Height
    End If

    ' Is the strikeout style available?
    If fontFamilies(j).IsStyleAvailable(FontStyle.Strikeout) Then
        familyNameAndStyle = ""
        familyNameAndStyle = familyNameAndStyle & familyName
        familyNameAndStyle = familyNameAndStyle & " Strikeout"
```

```
            Dim strikeFont As New Font( _
               familyName, _
               16, _
               FontStyle.Strikeout, _
               GraphicsUnit.Pixel)

            e.Graphics.DrawString( _
               familyNameAndStyle, _
               strikeFont, _
               solidBrush, _
               pointF)

            pointF.Y += strikeFont.Height
        End If

        ' Separate the families with white space.
        pointF.Y += 10
    End While
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

PrivateFontCollection
Using Fonts and Text

How to: Obtain Font Metrics

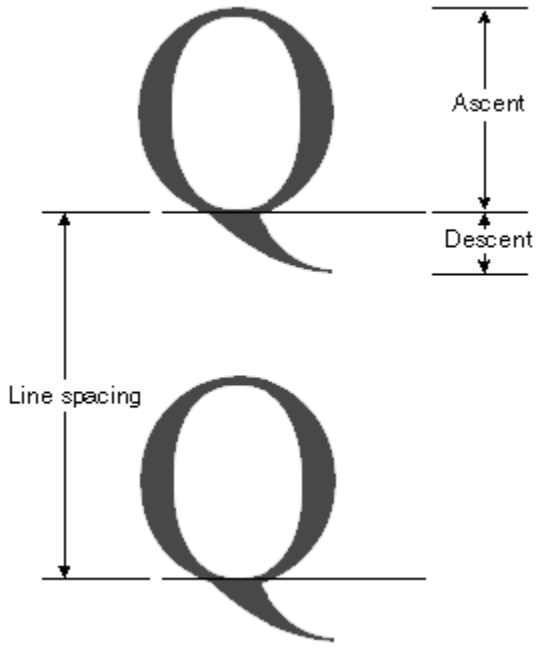
.NET Framework (current version)

The FontFamily class provides the following methods that retrieve various metrics for a particular family/style combination:

- GetEmHeight(FontStyle)
- GetCellAscent(FontStyle)
- GetCellDescent(FontStyle)
- GetLineSpacing(FontStyle)

The numbers returned by these methods are in font design units, so they are independent of the size and units of a particular Font object.

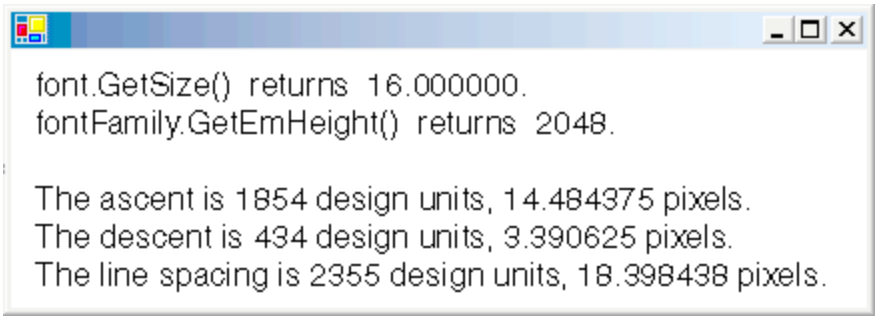
The following illustration shows the various metrics.



Example

The following example displays the metrics for the regular style of the Arial font family. The code also creates a Font object (based on the Arial family) with size 16 pixels and displays the metrics (in pixels) for that particular Font object.

The following illustration shows the output of the example code.



Note the first two lines of output in the preceding illustration. The Font object returns a size of 16, and the FontFamily object returns an em height of 2,048. These two numbers (16 and 2,048) are the key to converting between font design units and the units (in this case pixels) of the Font object.

For example, you can convert the ascent from design units to pixels as follows:

$$\frac{1854 \text{ design units}}{1} \times \frac{16 \text{ pixels}}{2048 \text{ design units}} = 14.484375 \text{ pixels}$$

The following code positions text vertically by setting the Y data member of a PointF object. The y-coordinate is increased by `font.Height` for each new line of text. The Height property of a Font object returns the line spacing (in pixels) for that particular Font object. In this example, the number returned by Height is 19. Note that this is the same as the number (rounded up to an integer) obtained by converting the line-spacing metric to pixels.

Note that the em height (also called size or em size) is not the sum of the ascent and the descent. The sum of the ascent and the descent is called the cell height. The cell height minus the internal leading is equal to the em height. The cell height plus the external leading is equal to the line spacing.

VB

```
Dim infoString As String = "" ' enough space for one line of output
Dim ascent As Integer ' font family ascent in design units
Dim ascentPixel As Single ' ascent converted to pixels
Dim descent As Integer ' font family descent in design units
Dim descentPixel As Single ' descent converted to pixels
Dim lineSpacing As Integer ' font family line spacing in design units
Dim lineSpacingPixel As Single ' line spacing converted to pixels
Dim fontFamily As New FontFamily("Arial")
Dim font As New Font( _
   fontFamily, _
   16, _
   FontStyle.Regular, _
   GraphicsUnit.Pixel)
Dim pointF As New PointF(10, 10)
Dim solidBrush As New SolidBrush(Color.Black)

' Display the font size in pixels.
infoString = "font.Size returns " & font.Size.ToString() & "."
e.Graphics.DrawString(infoString, font, solidBrush, pointF)

' Move down one line.
pointF.Y += font.Height

' Display the font family em height in design units.
infoString = "fontFamily.GetEmHeight() returns " & _
```

```
    fontFamily.GetEmHeight(FontStyle.Regular) & "."
e.Graphics.DrawString(infoString, font, solidBrush, pointF)

' Move down two lines.
pointF.Y += 2 * font.Height

' Display the ascent in design units and pixels.
ascent = fontFamily.GetCellAscent(FontStyle.Regular)

' 14.484375 = 16.0 * 1854 / 2048
ascentPixel = _
    font.Size * ascent / fontFamily.GetEmHeight(FontStyle.Regular)
infoString = "The ascent is " & ascent & " design units, " & ascentPixel _
    & " pixels."
e.Graphics.DrawString(infoString, font, solidBrush, pointF)

' Move down one line.
pointF.Y += font.Height

' Display the descent in design units and pixels.
descent = fontFamily.GetCellDescent(FontStyle.Regular)

' 3.390625 = 16.0 * 434 / 2048
descentPixel = _
    font.Size * descent / fontFamily.GetEmHeight(FontStyle.Regular)
infoString = "The descent is " & descent & " design units, " & _
    descentPixel & " pixels."
e.Graphics.DrawString(infoString, font, solidBrush, pointF)

' Move down one line.
pointF.Y += font.Height

' Display the line spacing in design units and pixels.
lineSpacing = fontFamily.GetLineSpacing(FontStyle.Regular)

' 18.398438 = 16.0 * 2355 / 2048
lineSpacingPixel = _
    font.Size * lineSpacing / fontFamily.GetEmHeight(FontStyle.Regular)
infoString = "The line spacing is " & lineSpacing & " design units, " & _
    lineSpacingPixel & " pixels."
e.Graphics.DrawString(infoString, font, solidBrush, pointF)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Graphics and Drawing in Windows Forms

Using Fonts and Text

How to: Use Antialiasing with Text

.NET Framework (current version)

Antialiasing refers to the smoothing of jagged edges of drawn graphics and text to improve their appearance or readability. With the managed GDI+ classes, you can render high quality antialiased text, as well as lower quality text. Typically, higher quality rendering takes more processing time than lower quality rendering. To set the text quality level, set the TextRenderingHint property of a Graphics to one of the elements of the TextRenderingHint enumeration

Example

The following code example draws text with two different quality settings.

The following illustration shows the output of the cod example code.

VB

```
Dim fontFamily As New FontFamily("Times New Roman")
Dim font As New Font( _
   fontFamily, _
   32, _
   FontStyle.Regular, _
   GraphicsUnit.Pixel)
Dim solidBrush As New SolidBrush(Color.FromArgb(255, 0, 0, 255))
Dim string1 As String = "SingleBitPerPixel"
Dim string2 As String = "AntiAlias"

e.Graphics.TextRenderingHint = TextRenderingHint.SingleBitPerPixel
e.Graphics.DrawString(string1, font, solidBrush, New PointF(10, 10))

e.Graphics.TextRenderingHint = TextRenderingHint.AntiAlias
e.Graphics.DrawString(string2, font, solidBrush, New PointF(10, 60))
```

Compiling the Code

The preceding code example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Using Fonts and Text

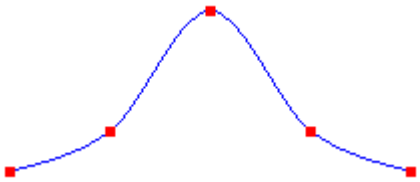
How to: Draw Cardinal Splines

.NET Framework (current version)

A cardinal spline is a curve that passes smoothly through a given set of points. To draw a cardinal spline, create a Graphics object and pass the address of an array of points to the DrawCurve method.

Drawing a Bell-Shaped Cardinal Spline

- The following example draws a bell-shaped cardinal spline that passes through five designated points. The following illustration shows the curve and five points.

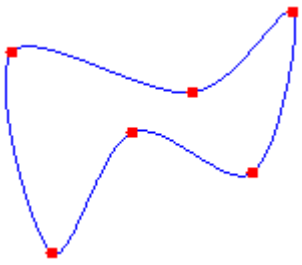


VB

```
Dim points As Point() = { _
   New Point(0, 100), _
   New Point(50, 80), _
   New Point(100, 20), _
   New Point(150, 80), _
   New Point(200, 100)}

Dim pen As New Pen(Color.FromArgb(255, 0, 0, 255))
e.Graphics.DrawCurve(pen, points)
```

Drawing a Closed Cardinal Spline

- Use the DrawClosedCurve method of the Graphics class to draw a closed cardinal spline. In a closed cardinal spline, the curve continues through the last point in the array and connects with the first point in the array. The following example draws a closed cardinal spline that passes through six designated points. The following illustration shows the closed spline along with the six points.

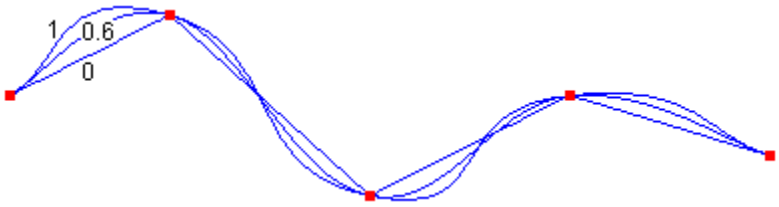


VB

```
Dim points As Point() = { _
   New Point(60, 60), _
   New Point(150, 80), _
   New Point(200, 40), _
   New Point(180, 120), _
   New Point(120, 100), _
   New Point(80, 160)}

Dim pen As New Pen(Color.FromArgb(255, 0, 0, 255))
e.Graphics.DrawClosedCurve(pen, points)
```

Changing the Bend of a Cardinal Spline

- Change the way a cardinal spline bends by passing a tension argument to the DrawCurve method. The following example draws three cardinal splines that pass through the same set of points. The following illustration shows the three splines along with their tension values. Note that when the tension is 0, the points are connected by straight lines.



VB

```
Dim points As Point() = { _
   New Point(20, 50), _
   New Point(100, 10), _
   New Point(200, 100), _
   New Point(300, 50), _
   New Point(400, 80)}

Dim pen As New Pen(Color.FromArgb(255, 0, 0, 255))
e.Graphics.DrawCurve(pen, points, 0.0F)
e.Graphics.DrawCurve(pen, points, 0.6F)
e.Graphics.DrawCurve(pen, points, 1.0F)
```

Compiling the Code

The preceding examples are designed for use with Windows Forms, and they require PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Lines, Curves, and Shapes
Constructing and Drawing Curves

How to: Draw a Single Bézier Spline

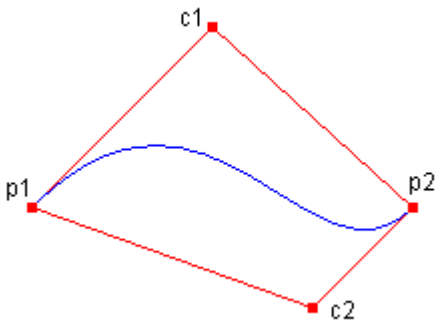
.NET Framework (current version)

A Bézier spline is defined by four points: a start point, two control points, and an endpoint.

Example

The following example draws a Bézier spline with start point (10, 100) and endpoint (200, 100). The control points are (100, 10) and (150, 150).

The following illustration shows the resulting Bézier spline along with its start point, control points, and endpoint. The illustration also shows the spline's convex hull, which is a polygon formed by connecting the four points with straight lines.



VB

```
Dim p1 As New Point(10, 100) ' Start point
Dim c1 As New Point(100, 10) ' First control point
Dim c2 As New Point(150, 150) ' Second control point
Dim p2 As New Point(200, 100) ' Endpoint

Dim pen As New Pen(Color.FromArgb(255, 0, 0, 255))
e.Graphics.DrawBezier(pen, p1, c1, c2, p2)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

DrawBezier
Bézier Splines in GDI+
How to: Draw a Sequence of Bézier Splines

How to: Draw a Sequence of Bézier Splines

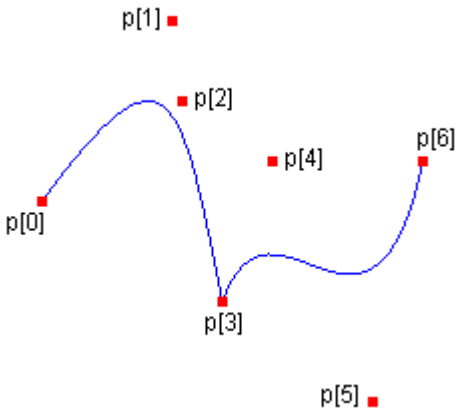
.NET Framework (current version)

You can use the DrawBeziers method of the Graphics class to draw a sequence of connected Bézier splines.

Example

The following example draws a curve that consists of two connected Bézier splines. The endpoint of the first Bézier spline is the start point of the second Bézier spline.

The following illustration shows the connected splines along with the seven points.



VB

```
' Point(10, 100) = start point of first spline
' Point(75, 10) = first control point of first spline
' Point(80, 50) = second control point of first spline

' Point(100, 150) = endpoint of first spline and start point of second spline

' Point(125, 80) = first control point of second spline
' Point(175, 200) = second control point of second spline
' Point(200, 80)} = endpoint of second spline
Dim p As Point() = { _
       New Point(10, 100), _
       New Point(75, 10), _
       New Point(80, 50), _
       New Point(100, 150), _
       New Point(125, 80), _
       New Point(175, 200), _
       New Point(200, 80)}

Dim pen As New Pen(Color.Blue)
e.Graphics.DrawBeziers(pen, p)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

- Graphics and Drawing in Windows Forms
- Bézier Splines in GDI+
- Constructing and Drawing Curves

How to: Create Figures from Lines, Curves, and Shapes

.NET Framework (current version)

To create a figure, construct a GraphicsPath, and then call methods, such as AddLine and AddCurve, to add primitives to the path.

Example

The following code examples create paths that have figures:

- The first example creates a path that has a single figure. The figure consists of a single arc. The arc has a sweep angle of –180 degrees, which is counterclockwise in the default coordinate system.
- The second example creates a path that has two figures. The first figure is an arc followed by a line. The second figure is a line followed by a curve followed by a line. The first figure is left open, and the second figure is closed.

VB

```
Dim path As New GraphicsPath()
path.AddArc(175, 50, 50, 50, 0, -180)
e.Graphics.DrawPath(New Pen(Color.FromArgb(128, 255, 0, 0), 4), path)
```

VB

```
' Create an array of points for the curve in the second figure.
Dim points As Point() = { _
   New Point(40, 60), _
   New Point(50, 70), _
   New Point(30, 90)}

Dim path As New GraphicsPath()

path.StartFigure() ' Start the first figure.
path.AddArc(175, 50, 50, 50, 0, -180)
path.AddLine(100, 0, 250, 20)
' First figure is not closed.

path.StartFigure() ' Start the second figure.
path.AddLine(50, 20, 5, 90)
path.AddCurve(points, 3)
path.AddLine(50, 150, 150, 180)
path.CloseFigure() ' Second figure is closed.
e.Graphics.DrawPath(New Pen(Color.FromArgb(255, 255, 0, 0), 2), path)
```

Compiling the Code

The previous examples are designed for use with Windows Forms, and they require PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

GraphicsPath
Constructing and Drawing Paths
Using a Pen to Draw Lines and Shapes

How to: Fill Open Figures

.NET Framework (current version)

You can fill a path by passing a GraphicsPath object to the FillPath method. The FillPath method fills the path according to the fill mode (alternate or winding) currently set for the path. If the path has any open figures, the path is filled as if those figures were closed. GDI+ closes a figure by drawing a straight line from its ending point to its starting point.

Example

The following example creates a path that has one open figure (an arc) and one closed figure (an ellipse). The FillPath method fills the path according to the default fill mode, which is Alternate.

The following illustration shows the output of the example code. Note that the path is filled (according to Alternate) as if the open figure were closed by a straight line from its ending point to its starting point.

VB

```
Dim path As New GraphicsPath()

' Add an open figure.
path.AddArc(0, 0, 150, 120, 30, 120)

' Add an intrinsically closed figure.
path.AddEllipse(50, 50, 50, 100)

Dim pen As New Pen(Color.FromArgb(128, 0, 0, 255), 5)
Dim brush As New SolidBrush(Color.Red)

' The fill mode is FillMode.Alternate by default.
e.Graphics.FillPath(brush, path)
e.Graphics.DrawPath(pen, path)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

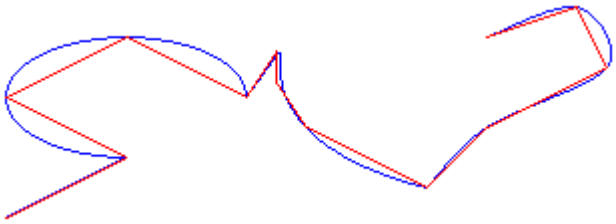
See Also

GraphicsPath
Graphics Paths in GDI+

How to: Flatten a Curved Path into a Line

.NET Framework (current version)

A GraphicsPath object stores a sequence of lines and Bézier splines. You can add several types of curves (ellipses, arcs, cardinal splines) to a path, but each curve is converted to a Bézier spline before it is stored in the path. Flattening a path consists of converting each Bézier spline in the path to a sequence of straight lines. The following illustration shows a path before and after flattening.



To Flatten a Path

- call the Flatten method of a GraphicsPath object. The Flatten method receives a flatness argument that specifies the maximum distance between the flattened path and the original path.

See Also

System.Drawing.Drawing2D.GraphicsPath
Lines, Curves, and Shapes
Constructing and Drawing Paths

Using the World Transformation

.NET Framework (current version)

The world transformation is a property of the Graphics class. The numbers that specify the world transformation are stored in a Matrix object, which represents a 3×3 matrix. The Matrix and Graphics classes have several methods for setting the numbers in the world transformation matrix.

Different Types of Transformations

In the following example, the code first creates a 50×50 rectangle and locates it at the origin (0, 0). The origin is at the upper-left corner of the client area.

VB

```
Dim rect As New Rectangle(0, 0, 50, 50)
Dim pen As New Pen(Color.FromArgb(128, 200, 0, 200), 2)
e.Graphics.DrawRectangle(pen, rect)
```

The following code applies a scaling transformation that expands the rectangle by a factor of 1.75 in the x direction and shrinks the rectangle by a factor of 0.5 in the y direction:

VB

```
e.Graphics.ScaleTransform(1.75F, 0.5F)
e.Graphics.DrawRectangle(pen, rect)
```

The result is a rectangle that is longer in the x direction and shorter in the y direction than the original.

To rotate the rectangle instead of scaling it, use the following code:

VB

```
e.Graphics.ResetTransform()
e.Graphics.RotateTransform(28) ' 28 degrees
e.Graphics.DrawRectangle(pen, rect)
```

To translate the rectangle, use the following code:

VB

```
e.Graphics.ResetTransform()
e.Graphics.TranslateTransform(150, 150)
e.Graphics.DrawRectangle(pen, rect)
```

See Also

Matrix
Coordinate Systems and Transformations
Using Transformations in Managed GDI+

Managing the State of a Graphics Object

.NET Framework (current version)

The Graphics class is at the heart of GDI+. To draw anything, you obtain a Graphics object, set its properties, and call its methods DrawLine, DrawImage, DrawString, and the like).

The following example calls the DrawRectangle method of a Graphics object. The first argument passed to the DrawRectangle method is a Pen object.

VB

```
Dim graphics As Graphics = e.Graphics
Dim pen As New Pen(Color.Blue) ' Opaque blue
graphics.DrawRectangle(pen, 10, 10, 200, 100)
```

Graphics State

A Graphics object does more than provide drawing methods, such as DrawLine and DrawRectangle. A Graphics object also maintains graphics state, which can be divided into the following categories:

- Quality settings
- Transformations
- Clipping region

Quality Settings

A Graphics object has several properties that influence the quality of the items that are drawn. For example, you can set the TextRenderingHint property to specify the type of antialiasing (if any) applied to text. Other properties that influence quality are SmoothingMode, CompositingMode, CompositingQuality, and InterpolationMode.

The following example draws two ellipses, one with the smoothing mode set to AntiAlias and one with the smoothing mode set to HighSpeed:

VB

```
Dim graphics As Graphics = e.Graphics
Dim pen As New Pen(Color.Blue)

graphics.SmoothingMode = SmoothingMode.AntiAlias
graphics.DrawEllipse(pen, 0, 0, 200, 100)
graphics.SmoothingMode = SmoothingMode.HighSpeed
graphics.DrawEllipse(pen, 0, 150, 200, 100)
```

Transformations

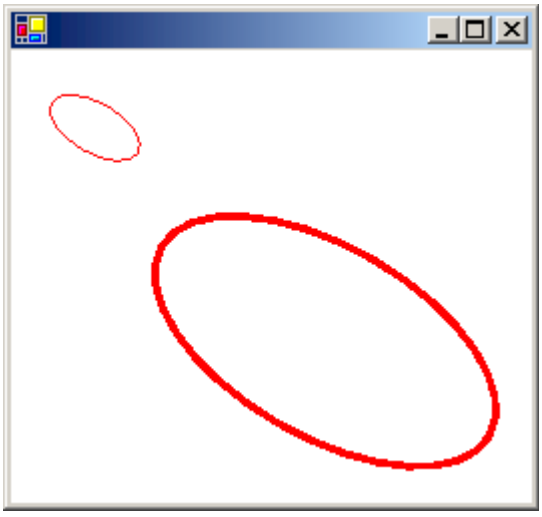
A Graphics object maintains two transformations (world and page) that are applied to all items drawn by that Graphics object. Any affine transformation can be stored in the world transformation. Affine transformations include scaling, rotating, reflecting, skewing, and translating. The page transformation can be used for scaling and for changing units (for example, pixels to inches). For more information, see Coordinate Systems and Transformations.

The following example sets the world and page transformations of a Graphics object. The world transformation is set to a 30-degree rotation. The page transformation is set so that the coordinates passed to the second DrawEllipse will be treated as millimeters instead of pixels. The code makes two identical calls to the DrawEllipse method. The world transformation is applied to the first DrawEllipse call, and both transformations (world and page) are applied to the second DrawEllipse call.

VB

```
Dim graphics As Graphics = e.Graphics
Dim pen As New Pen(Color.Red)

graphics.ResetTransform()
graphics.RotateTransform(30) ' world transformation
graphics.DrawEllipse(pen, 0, 0, 100, 50)
graphics.PageUnit = GraphicsUnit.Millimeter ' page transformation
graphics.DrawEllipse(pen, 0, 0, 100, 50)
```

The following illustration shows the two ellipses. Note that the 30-degree rotation is about the origin of the coordinate system (upper-left corner of the client area), not about the centers of the ellipses. Also note that the pen width of 1 means 1 pixel for the first ellipse and 1 millimeter for the second ellipse.



Clipping Region

A Graphics object maintains a clipping region that applies to all items drawn by that Graphics object. You can set the

clipping region by calling the SetClip method.

The following example creates a plus-shaped region by forming the union of two rectangles. That region is designated as the clipping region of a Graphics object. Then the code draws two lines that are restricted to the interior of the clipping region.

VB

```
Dim graphics As Graphics = e.Graphics

' Opaque red, width 5
Dim pen As New Pen(Color.Red, 5)

' Opaque aqua
Dim brush As New SolidBrush(Color.FromArgb(255, 180, 255, 255))

' Create a plus-shaped region by forming the union of two rectangles.
Dim [region] As New [Region](New Rectangle(50, 0, 50, 150))
[region].Union(New Rectangle(0, 50, 150, 50))
graphics.FillRegion(brush, [region])

' Set the clipping region.
graphics.SetClip([region], CombineMode.Replace)

' Draw two clipped lines.
graphics.DrawLine(pen, 0, 30, 150, 160)
graphics.DrawLine(pen, 40, 20, 190, 150)
```

The following illustration shows the clipped lines.

See Also

Graphics and Drawing in Windows Forms
Using Nested Graphics Containers

Why Transformation Order Is Significant

.NET Framework (current version)

A single Matrix object can store a single transformation or a sequence of transformations. The latter is called a composite transformation. The matrix of a composite transformation is obtained by multiplying the matrices of individual transformations.

Composite Transform Examples

In a composite transformation, the order of individual transformations is important. For example, if you first rotate, then scale, then translate, you get a different result than if you first translate, then rotate, then scale. In GDI+, composite transformations are built from left to right. If S, R, and T are scale, rotation, and translation matrices respectively, then the product SRT (in that order) is the matrix of the composite transformation that first scales, then rotates, then translates. The matrix produced by the product SRT is different from the matrix produced by the product TRS.

One reason order is significant is that transformations like rotation and scaling are done with respect to the origin of the coordinate system. Scaling an object that is centered at the origin produces a different result than scaling an object that has been moved away from the origin. Similarly, rotating an object that is centered at the origin produces a different result than rotating an object that has been moved away from the origin.

The following example combines scaling, rotation and translation (in that order) to form a composite transformation. The argument Append passed to the RotateTransform method indicates that the rotation will follow the scaling. Likewise, the argument Append passed to the TranslateTransform method indicates that the translation will follow the rotation. Append and Prepend are members of the MatrixOrder enumeration.

VB

```
Dim rect As New Rectangle(0, 0, 50, 50)
Dim pen As New Pen(Color.FromArgb(128, 200, 0, 200), 2)
e.Graphics.ResetTransform()
e.Graphics.ScaleTransform(1.75F, 0.5F)
e.Graphics.RotateTransform(28, MatrixOrder.Append)
e.Graphics.TranslateTransform(150, 150, MatrixOrder.Append)
e.Graphics.DrawRectangle(pen, rect)
```

The following example makes the same method calls as the preceding example, but the order of the calls is reversed. The resulting order of operations is first translate, then rotate, then scale, which produces a very different result than first scale, then rotate, then translate.

VB

```
Dim rect As New Rectangle(0, 0, 50, 50)
Dim pen As New Pen(Color.FromArgb(128, 200, 0, 200), 2)
e.Graphics.ResetTransform()
e.Graphics.TranslateTransform(150, 150, MatrixOrder.Append)
e.Graphics.RotateTransform(28, MatrixOrder.Append)
```

```
e.Graphics.ScaleTransform(1.75F, 0.5F)
e.Graphics.DrawRectangle(pen, rect)
```

One way to reverse the order of individual transformations in a composite transformation is to reverse the order of a sequence of method calls. A second way to control the order of operations is to change the matrix order argument. The following example is the same as the preceding example, except that Append has been changed to Prepend. The matrix multiplication is done in the order SRT, where S, R, and T are the matrices for scale, rotate, and translate, respectively. The order of the composite transformation is first scale, then rotate, then translate.

VB

```
Dim rect As New Rectangle(0, 0, 50, 50)
Dim pen As New Pen(Color.FromArgb(128, 200, 0, 200), 2)
e.Graphics.ResetTransform()
e.Graphics.TranslateTransform(150, 150, MatrixOrder.Prepend)
e.Graphics.RotateTransform(28, MatrixOrder.Prepend)
e.Graphics.ScaleTransform(1.75F, 0.5F)
e.Graphics.DrawRectangle(pen, rect)
```

The result of the immediately preceding example is the same as the result of the first example in this topic. This is because we reversed both the order of the method calls and the order of the matrix multiplication.

See Also

Matrix
Coordinate Systems and Transformations
Using Transformations in Managed GDI+

Using Nested Graphics Containers

.NET Framework (current version)

GDI+ provides containers that you can use to temporarily replace or augment part of the state in a Graphics object. You create a container by calling the BeginContainer method of a Graphics object. You can call BeginContainer repeatedly to form nested containers. Each call to BeginContainer must be paired with a call to EndContainer.

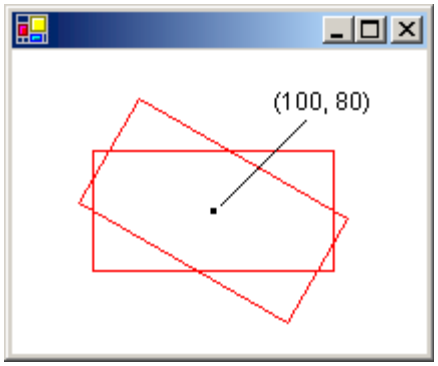
Transformations in Nested Containers

The following example creates a Graphics object and a container within that Graphics object. The world transformation of the Graphics object is a translation 100 units in the *x* direction and 80 units in the *y* direction. The world transformation of the container is a 30-degree rotation. The code makes the call `DrawRectangle(pen, -60, -30, 120, 60)` twice. The first call to DrawRectangle is inside the container; that is, the call is in between the calls to BeginContainer and EndContainer. The second call to DrawRectangle is after the call to EndContainer.

VB

```
Dim graphics As Graphics = e.Graphics
Dim pen As New Pen(Color.Red)
Dim graphicsContainer As GraphicsContainer
graphics.FillRectangle(Brushes.Black, 100, 80, 3, 3)

graphics.TranslateTransform(100, 80)

graphicsContainer = graphics.BeginContainer()
graphics.RotateTransform(30)
graphics.DrawRectangle(pen, -60, -30, 120, 60)
graphics.EndContainer(graphicsContainer)

graphics.DrawRectangle(pen, -60, -30, 120, 60)
```

In the preceding code, the rectangle drawn from inside the container is transformed first by the world transformation of the container (rotation) and then by the world transformation of the Graphics object (translation). The rectangle drawn from outside the container is transformed only by the world transformation of the Graphics object (translation). The following illustration shows the two rectangles.



Clipping in Nested Containers

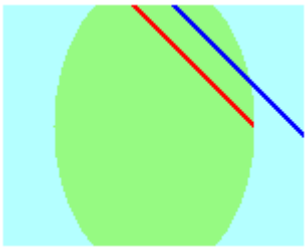
The following example demonstrates how nested containers handle clipping regions. The code creates a Graphics object and a container within that Graphics object. The clipping region of the Graphics object is a rectangle, and the clipping region of the container is an ellipse. The code makes two calls to the DrawLine method. The first call to DrawLine is inside the container, and the second call to DrawLine is outside the container (after the call to EndContainer). The first line is clipped by the intersection of the two clipping regions. The second line is clipped only by the rectangular clipping region of the Graphics object.

VB

```
Dim graphics As Graphics = e.Graphics
```

```
Dim graphicsContainer As GraphicsContainer
Dim redPen As New Pen(Color.Red, 2)
Dim bluePen As New Pen(Color.Blue, 2)
Dim aquaBrush As New SolidBrush(Color.FromArgb(255, 180, 255, 255))
Dim greenBrush As New SolidBrush(Color.FromArgb(255, 150, 250, 130))

graphics.SetClip(New Rectangle(50, 65, 150, 120))
graphics.FillRectangle(aquaBrush, 50, 65, 150, 120)

graphicsContainer = graphics.BeginContainer()
' Create a path that consists of a single ellipse.
Dim path As New GraphicsPath()
path.AddEllipse(75, 50, 100, 150)

' Construct a region based on the path.
Dim [region] As New [Region](path)
graphics.FillRegion(greenBrush, [region])

graphics.SetClip([region], CombineMode.Replace)
graphics.DrawLine(redPen, 50, 0, 350, 300)
graphics.EndContainer(graphicsContainer)

graphics.DrawLine(bluePen, 70, 0, 370, 300)
```

The following illustration shows the two clipped lines.



As the two preceding examples show, transformations and clipping regions are cumulative in nested containers. If you set the world transformations of the container and the Graphics object, both transformations will apply to items drawn from inside the container. The transformation of the container will be applied first, and the transformation of the Graphics object will be applied second. If you set the clipping regions of the container and the Graphics object, items drawn from inside the container will be clipped by the intersection of the two clipping regions.

Quality Settings in Nested Containers

Quality settings (SmoothingMode, TextRenderingHint, and the like) in nested containers are not cumulative; rather, the quality settings of the container temporarily replace the quality settings of a Graphics object. When you create a new container, the quality settings for that container are set to default values. For example, suppose you have a Graphics object with a smoothing mode of AntiAlias. When you create a container, the smoothing mode inside the container is the default smoothing mode. You are free to set the smoothing mode of the container, and any items drawn from inside the container will be drawn according to the mode you set. Items drawn after the call to EndContainer will be drawn according to the smoothing mode (AntiAlias) that was in place before the call to BeginContainer.

Several Layers of Nested Containers

You are not limited to one container in a Graphics object. You can create a sequence of containers, each nested in the preceding, and you can specify the world transformation, clipping region, and quality settings of each of those nested containers. If you call a drawing method from inside the innermost container, the transformations will be applied in order, starting with the innermost container and ending with the outermost container. Items drawn from inside the innermost container will be clipped by the intersection of all the clipping regions.

The following example creates a Graphics object and sets its text rendering hint to AntiAlias. The code creates two containers, one nested within the other. The text rendering hint of the outer container is set to SingleBitPerPixel, and the text rendering hint of the inner container is set to AntiAlias. The code draws three strings: one from the inner container, one from the outer container, and one from the Graphics object itself.

VB

```
Dim graphics As Graphics = e.Graphics
Dim innerContainer As GraphicsContainer
Dim outerContainer As GraphicsContainer
Dim brush As New SolidBrush(Color.Blue)
Dim fontFamily As New FontFamily("Times New Roman")
Dim font As New Font( _
   fontFamily, _
   36, _
   FontStyle.Regular, _
   GraphicsUnit.Pixel)

graphics.TextRenderingHint = _
System.Drawing.Text.TextRenderingHint.AntiAlias

outerContainer = graphics.BeginContainer()

graphics.TextRenderingHint = _
    System.Drawing.Text.TextRenderingHint.SingleBitPerPixel

innerContainer = graphics.BeginContainer()
graphics.TextRenderingHint = _
    System.Drawing.Text.TextRenderingHint.AntiAlias
graphics.DrawString( _
   "Inner Container", _
   font, _
   brush, _
   New PointF(20, 10))
graphics.EndContainer(innerContainer)

graphics.DrawString("Outer Container", font, brush, New PointF(20, 50))

graphics.EndContainer(outerContainer)

graphics.DrawString("Graphics Object", font, brush, New PointF(20, 90))
```

The following illustration shows the three strings. The strings drawn from the inner container and from the Graphics object

are smoothed by antialiasing. The string drawn from the outer container is not smoothed by antialiasing because the TextRenderingHint property is set to SingleBitPerPixel.

See Also

Graphics
Managing the State of a Graphics Object

How to: Use Hit Testing with a Region

.NET Framework (current version)

The purpose of hit testing is to determine whether the cursor is over a given object, such as an icon or a button.

Example

The following example creates a plus-shaped region by forming the union of two rectangular regions. Assume that the variable `point` holds the location of the most recent click. The code checks to see whether `point` is in the plus-shaped region. If the point is in the region (a hit), the region is filled with an opaque red brush. Otherwise, the region is filled with a semitransparent red brush.

VB

```
Dim point As New Point(60, 10)

' Assume that the variable "point" contains the location of the
' most recent mouse click.
' To simulate a hit, assign (60, 10) to point.
' To simulate a miss, assign (0, 0) to point.

Dim solidBrush As New SolidBrush(Color.Black)
Dim region1 As New [Region](New Rectangle(50, 0, 50, 150))
Dim region2 As New [Region](New Rectangle(0, 50, 150, 50))

' Create a plus-shaped region by forming the union of region1 and region2.
' The union replaces region1.
region1.Union(region2)

If region1.IsVisible(point, e.Graphics) Then
    ' The point is in the region. Use an opaque brush.
    solidBrush.Color = Color.FromArgb(255, 255, 0, 0)
Else
    ' The point is not in the region. Use a semitransparent brush.
    solidBrush.Color = Color.FromArgb(64, 255, 0, 0)
End If

e.Graphics.FillRegion(solidBrush, region1)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Region
Regions in GDI+
How to: Use Clipping with a Region

How to: Use Clipping with a Region

.NET Framework (current version)

One of the properties of the Graphics class is the clip region. All drawing done by a given Graphics object is restricted to the clip region of that Graphics object. You can set the clip region by calling the SetClip method.

Example

The following example constructs a path that consists of a single polygon. Then the code constructs a region, based on that path. The region is passed to the SetClip method of a Graphics object, and then two strings are drawn.

The following illustration shows the clipped strings.

VB

```
' Create a path that consists of a single polygon.
Dim polyPoints As Point() = { _
   New Point(10, 10), _
   New Point(150, 10), _
   New Point(100, 75), _
   New Point(100, 150)}
Dim path As New GraphicsPath()
path.AddPolygon(polyPoints)

' Construct a region based on the path.
Dim [region] As New [Region](path)

' Draw the outline of the region.
Dim pen As Pen = Pens.Black
e.Graphics.DrawPath(pen, path)

' Set the clipping region of the Graphics object.
e.Graphics.SetClip([region], CombineMode.Replace)

' Draw some clipped strings.
Dim fontFamily As New FontFamily("Arial")
Dim font As New Font( _
   fontFamily, _
   36, _
   FontStyle.Bold, _
```

```
    GraphicsUnit.Pixel)
Dim solidBrush As New SolidBrush(Color.FromArgb(255, 255, 0, 0))

e.Graphics.DrawString( _
    "A Clipping Region", _
    font, _
    solidBrush, _
    New PointF(15, 25))

e.Graphics.DrawString( _
    "A Clipping Region", _
    font, _
    solidBrush, _
    New PointF(15, 68))
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of PaintEventHandler.

See Also

Regions in GDI+
Using Regions

How to: Use a Color Matrix to Transform a Single Color

.NET Framework (current version)

GDI+ provides the Image and Bitmap classes for storing and manipulating images. Image and Bitmap objects store the color of each pixel as a 32-bit number: 8 bits each for red, green, blue, and alpha. Each of the four components is a number from 0 through 255, with 0 representing no intensity and 255 representing full intensity. The alpha component specifies the transparency of the color: 0 is fully transparent, and 255 is fully opaque.

A color vector is a 4-tuple of the form (red, green, blue, alpha). For example, the color vector (0, 255, 0, 255) represents an opaque color that has no red or blue, but has green at full intensity.

Another convention for representing colors uses the number 1 for full intensity. Using that convention, the color described in the preceding paragraph would be represented by the vector (0, 1, 0, 1). GDI+ uses the convention of 1 as full intensity when it performs color transformations.

You can apply linear transformations (rotation, scaling, and the like) to color vectors by multiplying the color vectors by a 4×4 matrix. However, you cannot use a 4×4 matrix to perform a translation (nonlinear). If you add a dummy fifth coordinate (for example, the number 1) to each of the color vectors, you can use a 5×5 matrix to apply any combination of linear transformations and translations. A transformation consisting of a linear transformation followed by a translation is called an affine transformation.

For example, suppose you want to start with the color (0.2, 0.0, 0.4, 1.0) and apply the following transformations:

1. Double the red component
2. Add 0.2 to the red, green, and blue components

The following matrix multiplication will perform the pair of transformations in the order listed.

$$\begin{bmatrix} 0.2 & 0.0 & 0.4 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0.2 & 0.2 & 0.2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.6 & 0.2 & 0.6 & 1.0 & 1.0 \end{bmatrix}$$

The elements of a color matrix are indexed (zero-based) by row and then column. For example, the entry in the fifth row and third column of matrix M is denoted by M[4][2].

The 5×5 identity matrix (shown in the following illustration) has 1s on the diagonal and 0s everywhere else. If you multiply a color vector by the identity matrix, the color vector does not change. A convenient way to form the matrix of a color transformation is to start with the identity matrix and make a small change that produces the desired transformation.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity Matrix

For a more detailed discussion of matrices and transformations, see Coordinate Systems and Transformations.

Example

The following example takes an image that is all one color (0.2, 0.0, 0.4, 1.0) and applies the transformation described in the preceding paragraphs.

The following illustration shows the original image on the left and the transformed image on the right.

The code in the following example uses the following steps to perform the recoloring:

1. Initialize a ColorMatrix object.
2. Create an ImageAttributes object and pass the ColorMatrix object to the SetColorMatrix method of the ImageAttributes object.
3. Pass the ImageAttributes object to the DrawImage method of a Graphics object.

VB

```
Dim image As New Bitmap("InputColor.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height

' The following matrix consists of the following transformations:
' red scaling factor of 2
' green scaling factor of 1
' blue scaling factor of 1
' alpha scaling factor of 1
' three translations of 0.2
Dim colorMatrixElements As Single()() = { _
   New Single() {2, 0, 0, 0, 0}, _
   New Single() {0, 1, 0, 0, 0}, _
   New Single() {0, 0, 1, 0, 0}, _
   New Single() {0, 0, 0, 1, 0}, _
   New Single() {0.2F, 0.2F, 0.2F, 0, 1}}

Dim colorMatrix As New ColorMatrix(colorMatrixElements)
```

```
imageAttributes.SetColorMatrix(colorMatrix, ColorMatrixFlag.Default,
ColorAdjustType.Bitmap)

e.Graphics.DrawImage(image, 10, 10)

e.Graphics.DrawImage( _
   image, _
   New Rectangle(120, 10, width, height), _
   0, _
   0, _
   width, _
   height, _
   GraphicsUnit.Pixel, _
   imageAttributes)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Recoloring Images
Coordinate Systems and Transformations

How to: Translate Image Colors

.NET Framework (current version)

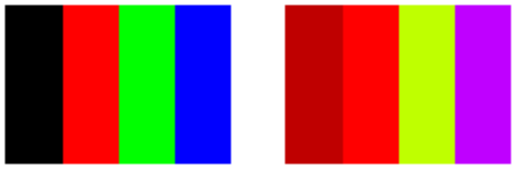
A translation adds a value to one or more of the four color components. The color matrix entries that represent translations are given in the following table.

Component to be translated	Matrix entry
Red	[4][0]
Green	[4][1]
Blue	[4][2]
Alpha	[4][3]

Example

The following example constructs an Image object from the file ColorBars.bmp. Then the code adds 0.75 to the red component of each pixel in the image. The original image is drawn alongside the transformed image.

The following illustration shows the original image on the left and the transformed image on the right.



The following table lists the color vectors for the four bars before and after the red translation. Note that because the maximum value for a color component is 1, the red component in the second row does not change. (Similarly, the minimum value for a color component is 0.)

Original	Translated
Black (0, 0, 0, 1)	(0.75, 0, 0, 1)
Red (1, 0, 0, 1)	(1, 0, 0, 1)
Green (0, 1, 0, 1)	(0.75, 1, 0, 1)
Blue (0, 0, 1, 1)	(0.75, 0, 1, 1)

VB

```
Dim image As New Bitmap("ColorBars.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height

Dim colorMatrixElements As Single()() = { _
   New Single() {1, 0, 0, 0, 0}, _
   New Single() {0, 1, 0, 0, 0}, _
   New Single() {0, 0, 1, 0, 0}, _
   New Single() {0, 0, 0, 1, 0}, _
   New Single() {0.75F, 0, 0, 0, 1}}

Dim colorMatrix As New ColorMatrix(colorMatrixElements)

imageAttributes.SetColorMatrix( _
   colorMatrix, _
   ColorMatrixFlag.Default, _
   ColorAdjustType.Bitmap)

e.Graphics.DrawImage(image, 10, 10, width, height)

' Pass in the destination rectangle (2nd argument), the upper-left corner
' (3rd and 4th arguments), width (5th argument),  and height (6th
' argument) of the source rectangle.
e.Graphics.DrawImage( _
   image, _
   New Rectangle(150, 10, width, height), _
   0, 0, _
   width, _
   height, _
   GraphicsUnit.Pixel, _
   imageAttributes)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Replace `ColorBars.bmp` with an image file name and path that are valid on your system.

See Also

- ColorMatrix
- ImageAttributes
- Graphics and Drawing in Windows Forms
- Recoloring Images

Using Transformations to Scale Colors

.NET Framework (current version)

A scaling transformation multiplies one or more of the four color components by a number. The color matrix entries that represent scaling are given in the following table.

Component to be scaled	Matrix entry
Red	[0][0]
Green	[1][1]
Blue	[2][2]
Alpha	[3][3]

Scaling One Color

The following example constructs an Image object from the file ColorBars2.bmp. Then the code scales the blue component of each pixel in the image by a factor of 2. The original image is drawn alongside the transformed image.

VB

```
Dim image As New Bitmap("ColorBars2.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height

Dim colorMatrixElements As Single()() = { _
   New Single() {1, 0, 0, 0, 0}, _
   New Single() {0, 1, 0, 0, 0}, _
   New Single() {0, 0, 2, 0, 0}, _
   New Single() {0, 0, 0, 1, 0}, _
   New Single() {0, 0, 0, 0, 1}}

Dim colorMatrix As New ColorMatrix(colorMatrixElements)

imageAttributes.SetColorMatrix( _
   colorMatrix, _
   ColorMatrixFlag.Default, _
   ColorAdjustType.Bitmap)

e.Graphics.DrawImage(image, 10, 10, width, height)
```

```
' Pass in the destination rectangle (2nd argument), the upper-left corner
' (3rd and 4th arguments), width (5th argument),  and height (6th
' argument) of the source rectangle.
e.Graphics.DrawImage( _
   image, _
   New Rectangle(150, 10, width, height), _
   0, 0, _
   width, _
   height, _
   GraphicsUnit.Pixel, _
   imageAttributes)
```

The following illustration shows the original image on the left and the scaled image on the right.

The following table lists the color vectors for the four bars before and after the blue scaling. Note that the blue component in the fourth color bar went from 0.8 to 0.6. That is because GDI+ retains only the fractional part of the result. For example, (2)(0.8) = 1.6, and the fractional part of 1.6 is 0.6. Retaining only the fractional part ensures that the result is always in the interval [0, 1].

Original	Scaled
(0.4, 0.4, 0.4, 1)	(0.4, 0.4, 0.8, 1)
(0.4, 0.2, 0.2, 1)	(0.4, 0.2, 0.4, 1)
(0.2, 0.4, 0.2, 1)	(0.2, 0.4, 0.4, 1)
(0.4, 0.4, 0.8, 1)	(0.4, 0.4, 0.6, 1)

Scaling Multiple Colors

The following example constructs an Image object from the file ColorBars2.bmp. Then the code scales the red, green, and blue components of each pixel in the image. The red components are scaled down 25 percent, the green components are scaled down 35 percent, and the blue components are scaled down 50 percent.

VB

```
Dim image As New Bitmap("ColorBars.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height
```

```
Dim colorMatrixElements As Single()() = { _
   New Single() {0.75F, 0, 0, 0, 0}, _
   New Single() {0, 0.65F, 0, 0, 0}, _
   New Single() {0, 0, 0.5F, 0, 0}, _
   New Single() {0, 0, 0, 1, 0}, _
   New Single() {0, 0, 0, 0, 1}}

Dim colorMatrix As New ColorMatrix(colorMatrixElements)

imageAttributes.SetColorMatrix( _
   colorMatrix, _
   ColorMatrixFlag.Default, _
   ColorAdjustType.Bitmap)

e.Graphics.DrawImage(image, 10, 10, width, height)

' Pass in the destination rectangle, and the upper-left corner, width,
' and height of the source rectangle as in the previous example.
e.Graphics.DrawImage( _
   image, _
   New Rectangle(150, 10, width, height), _
   0, 0, _
   width, _
   height, _
   GraphicsUnit.Pixel, _
   imageAttributes)
```

The following illustration shows the original image on the left and the scaled image on the right.

The following table lists the color vectors for the four bars before and after the red, green and blue scaling.

Original	**Scaled**
(0.6, 0.6, 0.6, 1)	(0.45, 0.39, 0.3, 1)
(0, 1, 1, 1)	(0, 0.65, 0.5, 1)
(1, 1, 0, 1)	(0.75, 0.65, 0, 1)
(1, 0, 1, 1)	(0.75, 0, 0.5, 1)

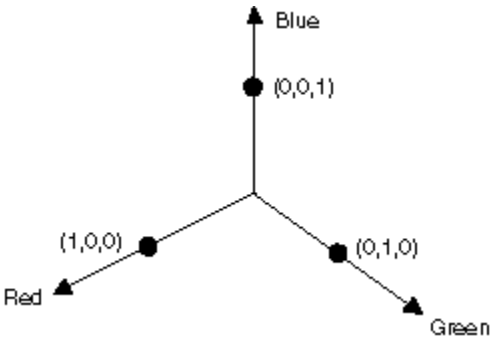
See Also

- ColorMatrix
- ImageAttributes
- Graphics and Drawing in Windows Forms
- Recoloring Images

How to: Rotate Colors

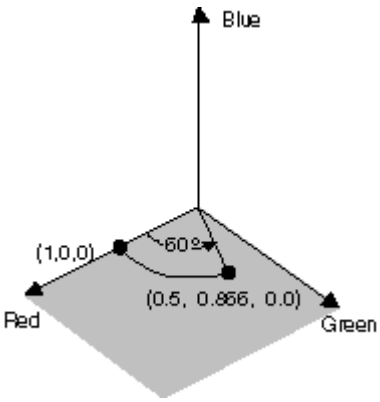
.NET Framework (current version)

Rotation in a four-dimensional color space is difficult to visualize. We can make it easier to visualize rotation by agreeing to keep one of the color components fixed. Suppose we agree to keep the alpha component fixed at 1 (fully opaque). Then we can visualize a three-dimensional color space with red, green, and blue axes as shown in the following illustration.

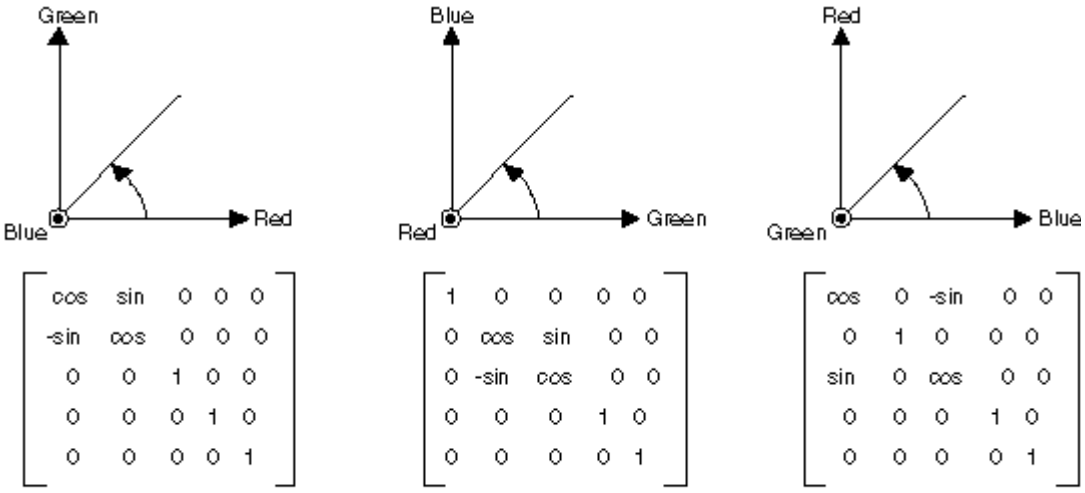


A color can be thought of as a point in 3-D space. For example, the point (1, 0, 0) in space represents the color red, and the point (0, 1, 0) in space represents the color green.

The following illustration shows what it means to rotate the color (1, 0, 0) through an angle of 60 degrees in the Red-Green plane. Rotation in a plane parallel to the Red-Green plane can be thought of as rotation about the blue axis.



The following illustration shows how to initialize a color matrix to perform rotations about each of the three coordinate axes (red, green, blue).



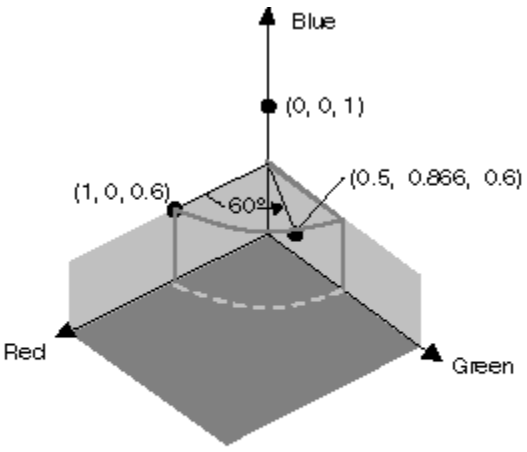
◉ Indicates that the axis comes out of the page toward the reader

Example

The following example takes an image that is all one color (1, 0, 0.6) and applies a 60-degree rotation about the blue axis. The angle of the rotation is swept out in a plane that is parallel to the red-green plane.

The following illustration shows the original image on the left and the color-rotated image on the right.

The following illustration shows a visualization of the color rotation performed in the following code.



The rotation takes place in the plane Blue=0.6, which is parallel to the Red-Green plane.

VB

```
Private Sub RotateColors(ByVal e As PaintEventArgs)
    Dim image As Bitmap = New Bitmap("RotationInput.bmp")
    Dim imageAttributes As New ImageAttributes()
    Dim width As Integer = image.Width
    Dim height As Integer = image.Height
```

```
        Dim degrees As Single = 60.0F
        Dim r As Double = degrees * System.Math.PI / 180 ' degrees to radians
        Dim colorMatrixElements As Single()() = { _
           New Single() {CSng(System.Math.Cos(r)), _
                         CSng(System.Math.Sin(r)), 0, 0, 0}, _
           New Single() {CSng(-System.Math.Sin(r)), _
                         CSng(-System.Math.Cos(r)), 0, 0, 0}, _
           New Single() {0, 0, 2, 0, 0}, _
           New Single() {0, 0, 0, 1, 0}, _
           New Single() {0, 0, 0, 0, 1}}

        Dim colorMatrix As New ColorMatrix(colorMatrixElements)

        imageAttributes.SetColorMatrix( _
           colorMatrix, _
           ColorMatrixFlag.Default, _
           ColorAdjustType.Bitmap)

        e.Graphics.DrawImage(image, 10, 10, width, height)

        ' Pass in the destination rectangle (2nd argument), the upper-left corner
        ' (3rd and 4th arguments), width (5th argument),  and height (6th
        ' argument) of the source rectangle.
        e.Graphics.DrawImage( _
           image, _
           New Rectangle(150, 10, width, height), _
           0, 0, _
           width, _
           height, _
           GraphicsUnit.Pixel, _
           imageAttributes)
    End Sub
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Replace *RotationInput.bmp* with an image file name and path valid on your system.

See Also

- ColorMatrix
- ImageAttributes
- Graphics and Drawing in Windows Forms
- Recoloring Images

How to: Shear Colors

.NET Framework (current version)

Shearing increases or decreases a color component by an amount proportional to another color component. For example, consider the transformation where the red component is increased by one half the value of the blue component. Under such a transformation, the color (0.2, 0.5, 1) would become (0.7, 0.5, 1). The new red component is 0.2 + (1/2)(1) = 0.7.

Example

The following example constructs an Image object from the file ColorBars4.bmp. Then the code applies the shearing transformation described in the preceding paragraph to each pixel in the image.

The following illustration shows the original image on the left and the sheared image on the right.

The following table lists the color vectors for the four bars before and after the shearing transformation.

Original	Sheared
(0, 0, 1, 1)	(0.5, 0, 1, 1)
(0.5, 1, 0.5, 1)	(0.75, 1, 0.5, 1)
(1, 1, 0, 1)	(1, 1, 0, 1)
(0.4, 0.4, 0.4, 1)	(0.6, 0.4, 0.4, 1)

VB

```
Dim image = New Bitmap("ColorBars.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height

Dim colorMatrixElements As Single()() = _
    {New Single() {1, 0, 0, 0, 0}, _
        New Single() {0, 1, 0, 0, 0}, _
        New Single() {0.5F, 0, 1, 0, 0}, _
        New Single() {0, 0, 0, 1, 0}, _
        New Single() {0, 0, 0, 0, 1}}
```

```
    Dim colorMatrix As New ColorMatrix(colorMatrixElements)

    imageAttributes.SetColorMatrix(colorMatrix, ColorMatrixFlag.Default, _
        ColorAdjustType.Bitmap)

    e.Graphics.DrawImage(image, 10, 10, width, height)

    e.Graphics.DrawImage(image, New Rectangle(150, 10, width, height), 0, 0, _
        width, height, GraphicsUnit.Pixel, imageAttributes)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler. Replace *ColorBars.bmp* with an image name and path valid on your system.

See Also

- ColorMatrix
- ImageAttributes
- Graphics and Drawing in Windows Forms
- Recoloring Images

How to: Use a Color Remap Table

.NET Framework (current version)

Remapping is the process of converting the colors in an image according to a color remap table. The color remap table is an array of ColorMap objects. Each ColorMap object in the array has an OldColor property and a NewColor property.

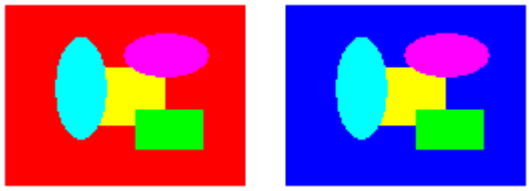
When GDI+ draws an image, each pixel of the image is compared to the array of old colors. If a pixel's color matches an old color, its color is changed to the corresponding new color. The colors are changed only for rendering — the color values of the image itself (stored in an Image or Bitmap object) are not changed.

To draw a remapped image, initialize an array of ColorMap objects. Pass that array to the SetRemapTable method of an ImageAttributes object, and then pass the ImageAttributes object to the DrawImage method of a Graphics object.

Example

The following example creates an Image object from the file RemapInput.bmp. The code creates a color remap table that consists of a single ColorMap object. The OldColor property of the `ColorRemap` object is red, and the NewColor property is blue. The image is drawn once without remapping and once with remapping. The remapping process changes all the red pixels to blue.

The following illustration shows the original image on the left and the remapped image on the right.



VB

```
Dim image As New Bitmap("RemapInput.bmp")
Dim imageAttributes As New ImageAttributes()
Dim width As Integer = image.Width
Dim height As Integer = image.Height
Dim colorMap As New ColorMap()

colorMap.OldColor = Color.FromArgb(255, 255, 0, 0) ' opaque red
colorMap.NewColor = Color.FromArgb(255, 0, 0, 255) ' opaque blue
Dim remapTable As ColorMap() = {colorMap}

imageAttributes.SetRemapTable(remapTable, ColorAdjustType.Bitmap)

e.Graphics.DrawImage(image, 10, 10, width, height)

' Pass in the destination rectangle (2nd argument), the upper-left corner
' (3rd and 4th arguments), width (5th argument),  and height (6th
' argument) of the source rectangle.
e.Graphics.DrawImage( _
```

```
    image, _
    New Rectangle(150, 10, width, height), _
    0, 0, _
    width, _
    height, _
    GraphicsUnit.Pixel, _
    imageAttributes)
```

Compiling the Code

The preceding example is designed for use with Windows Forms, and it requires PaintEventArgs *e*, which is a parameter of the Paint event handler.

See Also

Recoloring Images
Images, Bitmaps, and Metafiles

How to: List Installed Encoders

.NET Framework (current version)

You may want to list the image encoders available on a computer, to determine whether your application can save to a particular image file format. The ImageCodecInfo class provides the GetImageEncoders static methods so that you can determine which image encoders are available. GetImageEncoders returns an array of ImageCodecInfo objects.

Example

The following code example outputs the list of installed encoders and their property values.

VB

```
Private Sub GetImageEncodersExample(ByVal e As PaintEventArgs)
    ' Get an array of available encoders.
    Dim myCodecs() As ImageCodecInfo
    myCodecs = ImageCodecInfo.GetImageEncoders()
    Dim numCodecs As Integer = myCodecs.GetLength(0)

    ' Set up display variables.
    Dim foreColor As Color = Color.Black
    Dim font As New Font("Arial", 8)
    Dim i As Integer = 0

    ' Check to determine whether any codecs were found.
    If numCodecs > 0 Then

        ' Set up an array to hold codec information. There are 9
        ' information elements plus 1 space for each codec, so 10 times
        ' the number of codecs found is allocated.
        Dim myCodecInfo(numCodecs * 10) As String

        ' Write all the codec information to the array.
        For i = 0 To numCodecs - 1
            myCodecInfo((i * 10)) = "Codec Name = " + myCodecs(i).CodecName
            myCodecInfo((i * 10 + 1)) = "Class ID = " + myCodecs(i).Clsid.ToString()
            myCodecInfo((i * 10 + 2)) = "DLL Name = " + myCodecs(i).DllName
            myCodecInfo((i * 10 + 3)) = "Filename Ext. = " + myCodecs(i).FilenameExtension
            myCodecInfo((i * 10 + 4)) = "Flags = " + myCodecs(i).Flags.ToString()
            myCodecInfo((i * 10 + 5)) = "Format Descrip. = " +
myCodecs(i).FormatDescription
            myCodecInfo((i * 10 + 6)) = "Format ID = " + myCodecs(i).FormatID.ToString()
            myCodecInfo((i * 10 + 7)) = "MimeType = " + myCodecs(i).MimeType
            myCodecInfo((i * 10 + 8)) = "Version = " + myCodecs(i).Version.ToString()
            myCodecInfo((i * 10 + 9)) = " "
        Next i
        Dim numMyCodecInfo As Integer = myCodecInfo.GetLength(0)
```

```
            ' Render all of the information to the screen.
            Dim j As Integer = 20
            For i = 0 To numMyCodecInfo - 1
                e.Graphics.DrawString(myCodecInfo(i), _
                    font, New SolidBrush(foreColor), 20, j)
                j += 12
            Next i
        Else
            e.Graphics.DrawString("No Codecs Found", _
                font, New SolidBrush(foreColor), 20, 20)
        End If

    End Sub
```

Compiling the Code

This example requires:

- A Windows Forms application.
- A PaintEventArgs, which is a parameter of PaintEventHandler.

See Also

How to: List Installed Decoders
Using Image Encoders and Decoders in Managed GDI+

How to: List Installed Decoders

.NET Framework (current version)

You may want to list the image decoders available on a computer, to determine whether your application can read a particular image file format. The ImageCodecInfo class provides the GetImageDecoders static methods so that you can determine which image decoders are available. GetImageDecoders returns an array of ImageCodecInfo objects.

Example

The following code example outputs the list of installed decoders and their property values.

VB

```
Private Sub GetImageDecodersExample(ByVal e As PaintEventArgs)
    ' Get an array of available decoders.
    Dim myCodecs() As ImageCodecInfo
    myCodecs = ImageCodecInfo.GetImageDecoders()
    Dim numCodecs As Integer = myCodecs.GetLength(0)

    ' Set up display variables.
    Dim foreColor As Color = Color.Black
    Dim font As New Font("Arial", 8)
    Dim i As Integer = 0

    ' Check to determine whether any codecs were found.
    If numCodecs > 0 Then
        ' Set up an array to hold codec information. There are 9
        ' information elements plus 1 space for each codec, so 10 times
        ' the number of codecs found is allocated.
        Dim myCodecInfo(numCodecs * 10) As String

        ' Write all the codec information to the array.
        For i = 0 To numCodecs - 1
            myCodecInfo((i * 10)) = "Codec Name = " + myCodecs(i).CodecName
            myCodecInfo((i * 10 + 1)) = "Class ID = " + myCodecs(i).Clsid.ToString()
            myCodecInfo((i * 10 + 2)) = "DLL Name = " + myCodecs(i).DllName
            myCodecInfo((i * 10 + 3)) = "Filename Ext. = " + myCodecs(i).FilenameExtension
            myCodecInfo((i * 10 + 4)) = "Flags = " + myCodecs(i).Flags.ToString()
            myCodecInfo((i * 10 + 5)) = "Format Descrip. = " +
myCodecs(i).FormatDescription
            myCodecInfo((i * 10 + 6)) = "Format ID = " + myCodecs(i).FormatID.ToString()
            myCodecInfo((i * 10 + 7)) = "MimeType = " + myCodecs(i).MimeType
            myCodecInfo((i * 10 + 8)) = "Version = " + myCodecs(i).Version.ToString()
            myCodecInfo((i * 10 + 9)) = " "
        Next i
        Dim numMyCodecInfo As Integer = myCodecInfo.GetLength(0)

        ' Render all of the information to the screen.
```

```
            Dim j As Integer = 20
            For i = 0 To numMyCodecInfo - 1
                e.Graphics.DrawString(myCodecInfo(i), _
                    font, New SolidBrush(foreColor), 20, j)
                j += 12
            Next i
        Else
            e.Graphics.DrawString("No Codecs Found", _
                font, New SolidBrush(foreColor), 20, 20)
        End If
    End Sub
```

Compiling the Code

This example requires:

- A Windows Forms application.
- A PaintEventArgs, which is a parameter of PaintEventHandler.

See Also

How to: List Installed Encoders
Using Image Encoders and Decoders in Managed GDI+

How to: Determine the Parameters Supported by an Encoder

.NET Framework (current version)

You can adjust image parameters, such as quality and compression level, but you must know which parameters are supported by a given image encoder. The Image class provides the GetEncoderParameterList method so that you can determine which image parameters are supported for a particular encoder. You specify the encoder with a GUID. The GetEncoderParameterList method returns an array of EncoderParameter objects.

Example

The following example code outputs the supported parameters for the JPEG encoder. Use the list of parameter categories and associated GUIDs in the Encoder class overview to determine the category for each parameter.

VB

```
Private Sub GetSupportedParameters(ByVal e As PaintEventArgs)
    Dim bitmap1 As New Bitmap(1, 1)
    Dim jpgEncoder As ImageCodecInfo = GetEncoder(ImageFormat.Jpeg)
    Dim paramList As EncoderParameters = _
    bitmap1.GetEncoderParameterList(jpgEncoder.Clsid)
    Dim encParams As EncoderParameter() = paramList.Param
    Dim paramInfo As New StringBuilder()

    Dim i As Integer
    For i = 0 To encParams.Length - 1
        paramInfo.Append("Param " & i & " holds " & _
            encParams(i).NumberOfValues & " items of type " & _
            encParams(i).Type.ToString() & vbCr & vbLf & "Guid category: " & _
             encParams(i).Encoder.Guid.ToString() & vbCr & vbLf)
    Next i

    e.Graphics.DrawString(paramInfo.ToString(), _
       Me.Font, Brushes.Red, 10.0F, 10.0F)
End Sub

Private Function GetEncoder(ByVal format As ImageFormat) As ImageCodecInfo

    Dim codecs As ImageCodecInfo() = ImageCodecInfo.GetImageDecoders()

    Dim codec As ImageCodecInfo
    For Each codec In codecs
        If codec.FormatID = format.Guid Then
            Return codec
        End If
    Next codec
    Return Nothing
```

```
    End Function
```

Compiling the Code

This example requires:

- A Windows Forms application.
- A PaintEventArgs, which is a parameter of PaintEventHandler.

See Also

How to: List Installed Encoders
Types of Bitmaps
Using Image Encoders and Decoders in Managed GDI+

How to: Convert a BMP image to a PNG image

.NET Framework (current version)

Oftentimes, you will want to convert from one image file format to another. You can do this conversion easily by calling the Save method of the Image class and specifying the ImageFormat for the desired image file format.

Example

The following example loads a BMP image from a type, and saves the image in the PNG format.

VB

```
Private Sub SaveBmpAsPNG()
    Dim bmp1 As New Bitmap(GetType(Button), "Button.bmp")
    bmp1.Save("c:\button.png", ImageFormat.Png)

End Sub
```

Compiling the Code

This example requires:

- A Windows Forms application.
- A reference to the `System.Drawing.Imaging` namespace.

See Also

How to: List Installed Encoders
Using Image Encoders and Decoders in Managed GDI+
Types of Bitmaps

How to: Set JPEG Compression Level

.NET Framework (current version)

You may want to modify the parameters of an image when you save the image to disk to minimize the file size or improve its quality. You can adjust the quality of a JPEG image by modifying its compression level. To specify the compression level when you save a JPEG image, you must create an EncoderParameters object and pass it to the Save method of the Image class. Initialize the EncoderParameters object so that it has an array that consists of one EncoderParameter. When you create the EncoderParameter, specify the Quality encoder, and the desired compression level.

Example

The following example code creates an EncoderParameter object and saves three JPEG images. Each JPEG image is saved with a different quality level, by modifying the **long** value passed to the EncoderParameter constructor. A quality level of 0 corresponds to the greatest compression, and a quality level of 100 corresponds to the least compression.

VB

```
Private Sub VaryQualityLevel()
    ' Get a bitmap. The Using statement ensures objects
    ' are automatically disposed from memory after use.
    Using bmp1 As New Bitmap("C:\test\TestPhoto.jpg")
        Dim jpgEncoder As ImageCodecInfo = GetEncoder(ImageFormat.Jpeg)

        ' Create an Encoder object based on the GUID
        ' for the Quality parameter category.
        Dim myEncoder As System.Drawing.Imaging.Encoder =
System.Drawing.Imaging.Encoder.Quality

        ' Create an EncoderParameters object.
        ' An EncoderParameters object has an array of EncoderParameter
        ' objects. In this case, there is only one
        ' EncoderParameter object in the array.
        Dim myEncoderParameters As New EncoderParameters(1)

        Dim myEncoderParameter As New EncoderParameter(myEncoder, 50L)
        myEncoderParameters.Param(0) = myEncoderParameter
        bmp1.Save("c:\test\TestPhotoQualityFifty.jpg", jpgEncoder, myEncoderParameters)

        myEncoderParameter = New EncoderParameter(myEncoder, 100L)
        myEncoderParameters.Param(0) = myEncoderParameter
        bmp1.Save("C:\test\TestPhotoQualityHundred.jpg", jpgEncoder, myEncoderParameters)

        ' Save the bitmap as a JPG file with zero quality level compression.
        myEncoderParameter = New EncoderParameter(myEncoder, 0L)
        myEncoderParameters.Param(0) = myEncoderParameter
        bmp1.Save("C:\test\TestPhotoQualityZero.jpg", jpgEncoder, myEncoderParameters)
    End Using
End Sub
```

VB

```
Private Function GetEncoder(ByVal format As ImageFormat) As ImageCodecInfo

    Dim codecs As ImageCodecInfo() = ImageCodecInfo.GetImageDecoders()
    Dim codec As ImageCodecInfo
    For Each codec In codecs
        If codec.FormatID = format.Guid Then
            Return codec
        End If
    Next codec
    Return Nothing

End Function
```

Compiling the Code

This example requires:

- A Windows Forms application.
- A PaintEventArgs, which is a parameter of PaintEventHandler.
- An image file that is named `TestPhoto.jpg` and located at **c:**.

See Also

How to: Determine the Parameters Supported by an Encoder
Types of Bitmaps
Using Image Encoders and Decoders in Managed GDI+

Double Buffered Graphics

.NET Framework (current version)

Flicker is a common problem when programming graphics. Graphics operations that require multiple complex painting operations can cause the rendered images to appear to flicker or have an otherwise unacceptable appearance. To address these problems, the .NET Framework provides access to double buffering.

Double buffering uses a memory buffer to address the flicker problems associated with multiple paint operations. When double buffering is enabled, all paint operations are first rendered to a memory buffer instead of the drawing surface on the screen. After all paint operations are completed, the memory buffer is copied directly to the drawing surface associated with it. Because only one graphics operation is performed on the screen, the image flickering associated with complex painting operations is eliminated.

Default Double Buffering

The easiest way to use double buffering in your applications is to use the default double buffering for forms and controls that is provided by the .NET Framework. You can enable default double buffering for your Windows Forms and authored Windows controls by setting the DoubleBuffered property to **true** or by using the SetStyle method. For more information, see How to: Reduce Graphics Flicker with Double Buffering for Forms and Controls.

Manually Managing Buffered Graphics

For more advanced double buffering scenarios, such as animation or advanced memory management, you can use the .NET Framework classes to implement your own double-buffering logic. The class responsible for allocating and managing individual graphics buffers is the BufferedGraphicsContext class. Every application domain has its own default BufferedGraphicsContext instance that manages all of the default double buffering for that application. In most cases there will be only one application domain per application, so there is generally one default BufferedGraphicsContext per application. Default BufferedGraphicsContext instances are managed by the BufferedGraphicsManager class. You can retrieve a reference to the default BufferedGraphicsContext instance by calling the BufferedGraphicsManager.Current Property. You can also create a dedicated BufferedGraphicsContext instance, which can improve performance for graphically intensive applications. For information on how to create a BufferedGraphicsContext instance, see How to: Manually Manage Buffered Graphics.

Manually Displaying Buffered Graphics

You can use an instance of the BufferedGraphicsContext class to create graphics buffers by calling the BufferedGraphicsContext.Allocate Method, which returns an instance of the BufferedGraphics class. A BufferedGraphics object manages a memory buffer that is associated with a rendering surface, such as a form or control.

After it is instantiated, the BufferedGraphics class manages rendering to an in-memory graphics buffer. You can render graphics to the memory buffer through the BufferedGraphics.Graphics Property, which exposes a Graphics object that directly represents the memory buffer. You can paint to this Graphics object just as you would to a Graphics object that represents a drawing surface. After all the graphics have been drawn to the buffer, you can use the

BufferedGraphics.Render Method to copy the contents of the buffer to the drawing surface on the screen.

For more information on using the BufferedGraphics class, see Manually Rendering Buffered Graphics. For more information on rendering graphics, see Graphics and Drawing in Windows Forms

See Also

BufferedGraphics
BufferedGraphicsContext
BufferedGraphicsManager
How to: Manually Render Buffered Graphics
How to: Reduce Graphics Flicker with Double Buffering for Forms and Controls
How to: Manually Manage Buffered Graphics
Graphics and Drawing in Windows Forms

How to: Reduce Graphics Flicker with Double Buffering for Forms and Controls

.NET Framework (current version)

Double buffering uses a memory buffer to address the flicker problems associated with multiple paint operations. When double buffering is enabled, all paint operations are first rendered to a memory buffer instead of the drawing surface on the screen. After all paint operations are completed, the memory buffer is copied directly to the drawing surface associated with it. Because only one graphics operation is performed on the screen, the image flickering associated with complex painting operations is eliminated.For most applications, the default double buffering provided by the .NET Framework will provide the best results. Standard Windows Forms controls are double buffered by default. You can enable default double buffering in your forms and authored controls in two ways. You can either set the DoubleBuffered property to **true**, or you can call the SetStyle method to set the OptimizedDoubleBuffer flag to **true**. Both methods will enable default double buffering for your form or control and provide flicker-free graphics rendering. Calling the SetStyle method is recommended only for custom controls for which you have written all the rendering code.

For more advanced double buffering scenarios, such as animation or advanced memory management, you can implement your own double buffering logic. For more information, see How to: Manually Manage Buffered Graphics.

To reduce flicker

- Set the DoubleBuffered property to **true**.

VB

```
DoubleBuffered = True
```

- or -

- Call the SetStyle method to set the OptimizedDoubleBuffer flag to **true**.

VB

```
SetStyle(ControlStyles.OptimizedDoubleBuffer, True)
```

See Also

DoubleBuffered
SetStyle
Double Buffered Graphics
Graphics and Drawing in Windows Forms

How to: Manually Manage Buffered Graphics

.NET Framework (current version)

For more advanced double buffering scenarios, you can use the .NET Framework classes to implement your own double-buffering logic. The class responsible for allocating and managing individual graphics buffers is the BufferedGraphicsContext class. Every application has its own default BufferedGraphicsContext that manages all of the default double buffering for that application. You can retrieve a reference to this instance by calling the Current.

To obtain a reference to the default BufferedGraphicsContext

- Set the Current property, as shown in the following code example.

VB

```
Dim myContext As BufferedGraphicsContext
myContext = BufferedGraphicsManager.Current
```

Note

You do not need to call the **Dispose** method on the BufferedGraphicsContext reference that you receive from the BufferedGraphicsManager class. The BufferedGraphicsManager handles all of the memory allocation and distribution for default BufferedGraphicsContext instances.

For graphically intensive applications such as animation, you can sometimes improve performance by using a dedicated BufferedGraphicsContext instead of the BufferedGraphicsContext provided by the BufferedGraphicsManager. This enables you to create and manage graphics buffers individually, without incurring the performance overhead of managing all the other buffered graphics associated with your application, though the memory consumed by the application will be greater.

To create a dedicated BufferedGraphicsContext

- Declare and create a new instance of the BufferedGraphicsContext class, as shown in the following code example.

VB

```
Dim myContext As BufferedGraphicsContext
myContext = New BufferedGraphicsContext
' Insert code to create graphics here.
' On a nondefault BufferedGraphicsContext instance, you should always
' call Dispose when finished.
myContext.Dispose()
```

See Also

BufferedGraphicsContext
Double Buffered Graphics
How to: Manually Render Buffered Graphics

How to: Manually Render Buffered Graphics

.NET Framework (current version)

If you are managing your own buffered graphics, you will need to be able to create and render graphics buffers. You can create instances of the BufferedGraphics class that is associated with drawing surfaces on your screen by calling the Allocate method. This method creates a BufferedGraphics instance that is associated with a particular rendering surface, such as a form or control. After you have created a BufferedGraphics instance, you can draw graphics to the buffer it represents through the Graphics property. After you have performed all graphics operations, you can copy the contents of the buffer to the screen by calling the Render method.

Note

If you perform your own rendering, memory consumption will increase, though the increase may only be slight.

To manually display buffered graphics

1. Obtain a reference to an instance of the BufferedGraphicsContext class. For more information, see How to: Manually Manage Buffered Graphics.

2. Create an instance of the BufferedGraphics class by calling the Allocate method, as shown in the following code example.

VB

```
' This example assumes the existence of a form called Form1.
Dim currentContext As BufferedGraphicsContext
Dim myBuffer As BufferedGraphics
' Gets a reference to the current BufferedGraphicsContext.
currentContext = BufferedGraphicsManager.Current
' Creates a BufferedGraphics instance associated with Form1, and with
' dimensions the same size as the drawing surface of Form1.
myBuffer = currentContext.Allocate(Me.CreateGraphics, _
   Me.DisplayRectangle)
```

3. Draw graphics to the graphics buffer by setting the Graphics property. For example:

VB

```
' Draws an ellipse to the graphics buffer.
myBuffer.Graphics.DrawEllipse(Pens.Blue, Me.DisplayRectangle)
```

4. When you have completed all of your drawing operations to the graphics buffer, call the Render method to render

the buffer, either to the drawing surface associated with that buffer, or to a specified drawing surface, as shown in the following code example.

VB

```
' Renders the contents of the buffer to the drawing surface associated
' with the buffer.
myBuffer.Render()
' Renders the contents of the buffer to the specified drawing surface.
myBuffer.Render(Me.CreateGraphics)
```

5. After you are finished rendering graphics, call the **Dispose** method on the BufferedGraphics instance to free system resources.

VB

```
myBuffer.Dispose()
```

See Also

BufferedGraphicsContext
BufferedGraphics
Double Buffered Graphics
How to: Manually Manage Buffered Graphics