# Using Managed Graphics Classes

**.NET Framework (current version)**

The following topics describe how to use the GDI+ API in the managed class framework.

## In This Section

Getting Started with Graphics Programming
Describes how to accomplish basic tasks with GDI+.

Using a Pen to Draw Lines and Shapes
Demonstrates how to construct a pen and use it to draw a variety of lines and shapes.

Using a Brush to Fill Shapes
Demonstrates how to construct a brush and fill shapes with a variety of effects.

Using a Gradient Brush to Fill Shapes
Shows how to create and use different types of gradient brushes.

Working with Images, Bitmaps, Icons, and Metafiles
Demonstrates how to construct and manipulate images.

Alpha Blending Lines and Fills
Demonstrates how to achieve transparency for shapes and lines.

Using Fonts and Text
Shows how to draw text and use fonts and font families.

Constructing and Drawing Curves
Shows how to draw Cardinal and Bezier splines.

Constructing and Drawing Paths
Shows how to create figures using paths.

Using Transformations in Managed GDI+
Demonstrates matrix transformations.

Using Graphics Containers
Shows how to manage graphics object state and nested graphics containers.

Using Regions
Demonstrates hit testing and clipping with regions.

Recoloring Images
Demonstrates various aspects of manipulating colors.

Using Image Encoders and Decoders in Managed GDI+
Show how to use image encoders and decoders to manipulate images.

Double Buffered Graphics

Demonstrates how to reduce flicker with double buffering.

# Control.OnPaint Method (PaintEventArgs)

**.NET Framework (current version)**

Raises the Paint event.

**Namespace:** System.Windows.Forms
**Assembly:** System.Windows.Forms (in System.Windows.Forms.dll)

## Syntax

**VB**

```
Protected Overridable Sub OnPaint (
        e As PaintEventArgs
)
```

**Parameters**

*e*

Type: System.Windows.Forms.PaintEventArgs
A PaintEventArgs that contains the event data.

## Remarks

Raising an event invokes the event handler through a delegate. For more information, see Handling and Raising Events.

The OnPaint method also enables derived classes to handle the event without attaching a delegate. This is the preferred technique for handling the event in a derived class.

## Notes to Inheritors:

When overriding OnPaint in a derived class, be sure to call the base class's OnPaint method so that registered delegates receive the event.

## Examples

The following code example enables the user to drag an image or image file onto the form, and have it be displayed at the point on which it is dropped. The OnPaint method is overridden to repaint the image each time the form is painted; otherwise the image would only persist until the next repainting. The DragEnter event-handling method determines the type of data being dragged into the form and provides the appropriate feedback. The DragDrop event-handling method displays the image on the form, if an Image can be created from the data. Because the DragEventArgs.X and

DragEventArgs.Y values are screen coordinates, the example uses the PointToClient method to convert them to client coordinates.

**VB**

```
Private picture As Image
Private pictureLocation As Point

Public Sub New()
   ' Enable drag-and-drop operations.
   Me.AllowDrop = True
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
   MyBase.OnPaint(e)

   ' If there is an image and it has a location,
   ' paint it when the Form is repainted.
   If (Me.picture IsNot Nothing) And _
     Not (Me.pictureLocation.Equals(Point.Empty)) Then
      e.Graphics.DrawImage(Me.picture, Me.pictureLocation)
   End If
End Sub

Private Sub Form1_DragDrop(ByVal sender As Object, _
  ByVal e As DragEventArgs) Handles MyBase.DragDrop
   ' Handle FileDrop data.
   If e.Data.GetDataPresent(DataFormats.FileDrop) Then
      ' Assign the file names to a string array, in
      ' case the user has selected multiple files.
      Dim files As String() = CType(e.Data.GetData(DataFormats.FileDrop), String())
      Try
         ' Assign the first image to the 'picture' variable.
         Me.picture = Image.FromFile(files(0))
         ' Set the picture location equal to the drop point.
         Me.pictureLocation = Me.PointToClient(New Point(e.X, e.Y))
      Catch ex As Exception
         MessageBox.Show(ex.Message)
         Return
      End Try
   End If

   ' Handle Bitmap data.
   If e.Data.GetDataPresent(DataFormats.Bitmap) Then
      Try
         ' Create an Image and assign it to the picture variable.
         Me.picture = CType(e.Data.GetData(DataFormats.Bitmap), Image)
         ' Set the picture location equal to the drop point.
         Me.pictureLocation = Me.PointToClient(New Point(e.X, e.Y))
      Catch ex As Exception
         MessageBox.Show(ex.Message)
         Return
      End Try
   End If
```

```
    ' Force the form to be redrawn with the image.
    Me.Invalidate()
End Sub

Private Sub Form1_DragEnter(ByVal sender As Object, _
  ByVal e As DragEventArgs) Handles MyBase.DragEnter
    ' If the data is a file or a bitmap, display the copy cursor.
    If e.Data.GetDataPresent(DataFormats.Bitmap) _
       Or e.Data.GetDataPresent(DataFormats.FileDrop) Then
       e.Effect = DragDropEffects.Copy
    Else
       e.Effect = DragDropEffects.None
    End If
End Sub
```

**VB**

```
' This example creates a PictureBox control on the form and draws to it.
' This example assumes that the Form_Load event handler method is connected
' to the Load event of the form.
Private pictureBox1 As New PictureBox()

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    ' Dock the PictureBox to the form and set its background to white.
    pictureBox1.Dock = DockStyle.Fill
    pictureBox1.BackColor = Color.White
    ' Connect the Paint event of the PictureBox to the event handler method.
    AddHandler pictureBox1.Paint, AddressOf Me.pictureBox1_Paint

    ' Add the PictureBox control to the Form.
    Me.Controls.Add(pictureBox1)
End Sub 'Form1_Load


Private Sub pictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs)
    ' Create a local version of the graphics object for the PictureBox.
    Dim g As Graphics = e.Graphics

    ' Draw a string on the PictureBox.
    g.DrawString("This is a diagonal line drawn on the control", _
        New Font("Arial", 10), Brushes.Red, New PointF(30.0F, 30.0F))
    ' Draw a line in the PictureBox.
    g.DrawLine(System.Drawing.Pens.Red, pictureBox1.Left, _
        pictureBox1.Top, pictureBox1.Right, pictureBox1.Bottom)
End Sub 'pictureBox1_Paint
```

# Version Information

**.NET Framework**
Available since 1.1

## See Also

Paint
Control Class
System.Windows.Forms Namespace

Return to top

# Getting Started with Graphics Programming

**.NET Framework (current version)**

This section shows how to get started using GDI+ in a Windows Forms application. The following topics show how to complete several GDI+ tasks such as drawing and filling shapes and text.

## In This Section

How to: Create Graphics Objects for Drawing
Shows how to create a Graphics object for drawing.

How to: Create a Pen
Shows how to create a pen.

How to: Set the Color of a Pen
Demonstrates how to set the color of a pen.

How to: Create a Solid Brush
Describes how to create a solid brush.

How to: Draw a Line on a Windows Form
Demonstrates how to draw a line.

How to: Draw an Outlined Shape
Describes how to draw a shape.

How to: Draw a Filled Rectangle on a Windows Form
Explains how to draw a rectangle.

How to: Draw a Filled Ellipse on a Windows Form
Shows how to draw a filled ellipse.

How to: Draw Text on a Windows Form
Describes how to draw text.

How to: Draw Vertical Text on a Windows Form
Shows how to draw vertical text.

How to: Render Images with GDI+
Demonstrates how to draw images.

How to: Create a Shaped Windows Form
Explains how to change the shape of a form.

How to: Copy Pixels for Reducing Flicker in Windows Forms
Explains how to copy pixels from one area to another.

## Reference

System.Drawing

Describes this namespace and has links to all its members.

System.Windows.Forms

Describes this namespace and has links to all of its members.

# How to: Create Graphics Objects for Drawing

**.NET Framework (current version)**

Before you can draw lines and shapes, render text, or display and manipulate images with GDI+, you need to create a Graphics object. The Graphics object represents a GDI+ drawing surface, and is the object that is used to create graphical images.

There are two steps in working with graphics:

1. Creating a Graphics object.
2. Using the Graphics object to draw lines and shapes, render text, or display and manipulate images.

## Creating a Graphics Object

A graphics object can be created in a variety of ways.

### To create a graphics object

- Receive a reference to a graphics object as part of the PaintEventArgs in the Paint event of a form or control. This is usually how you obtain a reference to a graphics object when creating painting code for a control. Similarly, you can also obtain a graphics object as a property of the PrintPageEventArgs when handling the PrintPage event for a PrintDocument.

  -or-

- Call the CreateGraphics method of a control or form to obtain a reference to a Graphics object that represents the drawing surface of that control or form. Use this method if you want to draw on a form or control that already exists.

  -or-

- Create a Graphics object from any object that inherits from Image. This approach is useful when you want to alter an already existing image.

  The following sections give details about each of these processes.

## PaintEventArgs in the Paint Event Handler

When programming the PaintEventHandler for controls or the PrintPage for a PrintDocument, a graphics object is provided as one of the properties of PaintEventArgs or PrintPageEventArgs.

### To obtain a reference to a Graphics object from the PaintEventArgs in the Paint event

1. Declare the Graphics object.

2. Assign the variable to refer to the Graphics object passed as part of the PaintEventArgs.

3. Insert code to paint the form or control.

   The following example shows how to reference a Graphics object from the PaintEventArgs in the Paint event:

**VB**

```
Private Sub Form1_Paint(sender As Object, pe As PaintEventArgs) Handles _
   MyBase.Paint
   ' Declares the Graphics object and sets it to the Graphics object
   ' supplied in the PaintEventArgs.
   Dim g As Graphics = pe.Graphics
   ' Insert code to paint the form here.
End Sub
```

# CreateGraphics Method

You can also use the CreateGraphics method of a control or form to obtain a reference to a Graphics object that represents the drawing surface of that control or form.

### To create a Graphics object with the CreateGraphics method

- Call the CreateGraphics method of the form or control upon which you want to render graphics.

**VB**

```
Dim g as Graphics
' Sets g to a Graphics object representing the drawing surface of the
' control or form g is a member of.
g = Me.CreateGraphics
```

# Create from an Image Object

Additionally, you can create a graphics object from any object that derives from the Image class.

### To create a Graphics object from an Image

- Call the Graphics.FromImage method, supplying the name of the Image variable from which you want to create a Graphics object.

The following example shows how to use a Bitmap object:

**VB**

```
Dim myBitmap as New Bitmap("C:\Documents and Settings\Joe\Pics\myPic.bmp")
Dim g as Graphics = Graphics.FromImage(myBitmap)
```

**Note**

You can only create Graphics objects from nonindexed .bmp files, such as 16-bit, 24-bit, and 32-bit .bmp files. Each pixel of nonindexed .bmp files holds a color, in contrast to pixels of indexed .bmp files, which hold an index to a color table.

- 

# Drawing and Manipulating Shapes and Images

After it is created, a Graphics object may be used to draw lines and shapes, render text, or display and manipulate images. The principal objects that are used with the Graphics object are:

- The Pen class—Used for drawing lines, outlining shapes, or rendering other geometric representations.
- The Brush class—Used for filling areas of graphics, such as filled shapes, images, or text.
- The Font class—Provides a description of what shapes to use when rendering text.
- The Color structure—Represents the different colors to display.

### To use the Graphics object you have created

- Work with the appropriate object listed above to draw what you need.

  For more information, see the following topics:

| To render | See |
|---|---|
| Lines | How to: Draw a Line on a Windows Form |
| Shapes | How to: Draw an Outlined Shape |
| Text | How to: Draw Text on a Windows Form |
| Images | How to: Render Images with GDI+ |

## See Also

Getting Started with Graphics Programming
Graphics and Drawing in Windows Forms
Lines, Curves, and Shapes
How to: Render Images with GDI+

# How to: Create a Pen

**.NET Framework (current version)**

This example creates a Pen object.

## Example

**VB**

```
Dim myPen As System.Drawing.Pen
myPen = New System.Drawing.Pen(System.Drawing.Color.Tomato)
```

## Robust Programming

After you have finished using objects that consume system resources, such as Pen objects, you should call Dispose on them.

## See Also

Pen
Getting Started with Graphics Programming
Pens, Lines, and Rectangles in GDI+

# How to: Set the Color of a Pen

**.NET Framework (current version)**

This example changes the color of a pre-existing Pen object

## Example

**VB**

```
myPen.Color = System.Drawing.Color.PeachPuff
```

## Compiling the Code

This example requires:

- A Pen object named `myPen`.

## Robust Programming

You should call Dispose on objects that consume system resources (such as Pen objects) after you are finished using them.

## See Also

Pen
Getting Started with Graphics Programming
How to: Create a Pen
Using a Pen to Draw Lines and Shapes
Pens, Lines, and Rectangles in GDI+

# How to: Create a Solid Brush

**.NET Framework (current version)**

This example creates a SolidBrush object that can be used by a Graphics object for filling shapes.

## Example

**VB**

```
Dim myBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
Dim formGraphics As System.Drawing.Graphics
formGraphics = Me.CreateGraphics()
formGraphics.FillEllipse(myBrush, New Rectangle(0, 0, 200, 300))
myBrush.Dispose()
formGraphics.Dispose()
```

## Robust Programming

After you have finished using them, you should call Dispose on objects that consume system resources, such as brush objects.

## See Also

- SolidBrush
- Brush
- Getting Started with Graphics Programming
- Brushes and Filled Shapes in GDI+
- Using a Brush to Fill Shapes

# How to: Draw a Line on a Windows Form

**.NET Framework (current version)**

This example draws a line on a form. Typically, when you draw on a form, you handle the form's Paint event and perform the drawing using the Graphics property of the PaintEventArgs, as shown in this example

## Example

**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.

## Robust Programming

You should always call Dispose on any objects that consume system resources, such as Pen objects.

## See Also

- DrawLine
- OnPaint
- Getting Started with Graphics Programming
- Using a Pen to Draw Lines and Shapes
- Graphics and Drawing in Windows Forms

# How to: Draw an Outlined Shape

**.NET Framework (current version)**

This example draws outlined ellipses and rectangles on a form.

## Example

VB

```
Private Sub DrawEllipse()
    Dim myPen As New System.Drawing.Pen(System.Drawing.Color.Red)
    Dim formGraphics As System.Drawing.Graphics
    formGraphics = Me.CreateGraphics()
    formGraphics.DrawEllipse(myPen, New Rectangle(0, 0, 200, 300))
    myPen.Dispose()
    formGraphics.Dispose()
End Sub

Private Sub DrawRectangle()
    Dim myPen As New System.Drawing.Pen(System.Drawing.Color.Red)
    Dim formGraphics As System.Drawing.Graphics
    formGraphics = Me.CreateGraphics()
    formGraphics.DrawRectangle(myPen, New Rectangle(0, 0, 200, 300))
    myPen.Dispose()
    formGraphics.Dispose()
End Sub
```

## Compiling the Code

You cannot call this method in the Load event handler. The drawn content will not be redrawn if the form is resized or obscured by another form. To make your content automatically repaint, you should override the OnPaint method.

## Robust Programming

You should always call Dispose on any objects that consume system resources, such as Pen and Graphics objects.

## See Also

- DrawEllipse
- OnPaint
- DrawRectangle
- Getting Started with Graphics Programming
- Using a Pen to Draw Lines and Shapes
- Graphics and Drawing in Windows Forms

# How to: Draw a Filled Rectangle on a Windows Form

**.NET Framework (current version)**

This example draws a filled rectangle on a form.

## Example

**VB**

```
Dim myBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
Dim formGraphics As System.Drawing.Graphics
formGraphics = Me.CreateGraphics()
formGraphics.FillRectangle(myBrush, New Rectangle(0, 0, 200, 300))
myBrush.Dispose()
formGraphics.Dispose()
```

## Compiling the Code

You cannot call this method in the Load event handler. The drawn content will not be redrawn if the form is resized or obscured by another form. To make your content automatically repaint, you should override the OnPaint method.

## Robust Programming

You should always call Dispose on any objects that consume system resources, such as Brush and Graphics objects.

## See Also

FillRectangle
OnPaint
Getting Started with Graphics Programming
Graphics and Drawing in Windows Forms
Using a Pen to Draw Lines and Shapes
Brushes and Filled Shapes in GDI+

# How to: Draw a Filled Ellipse on a Windows Form

**.NET Framework (current version)**

This example draws a filled ellipse on a form.

## Example

**VB**

```
Dim myBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
Dim formGraphics As System.Drawing.Graphics
formGraphics = Me.CreateGraphics()
formGraphics.FillEllipse(myBrush, New Rectangle(0, 0, 200, 300))
myBrush.Dispose()
formGraphics.Dispose()
```

## Compiling the Code

You cannot call this method in the Load event handler. The drawn content will not be redrawn if the form is resized or obscured by another form. To make your content automatically repaint, you should override the OnPaint method.

## Robust Programming

You should always call Dispose on any objects that consume system resources, such as Brush and Graphics objects.

## See Also

- Graphics and Drawing in Windows Forms
- Getting Started with Graphics Programming
- Alpha Blending Lines and Fills
- Using a Brush to Fill Shapes

# How to: Draw Text on a Windows Form

**.NET Framework (current version)**

The following code example shows how to use the DrawString method of the Graphics to draw text on a form. Alternatively, you can use TextRenderer for drawing text on a form. For more information, see How to: Draw Text with GDI.

## Example

**VB**

```
Public Sub DrawString()
    Dim formGraphics As System.Drawing.Graphics = Me.CreateGraphics()
    Dim drawString As String = "Sample Text"
    Dim drawFont As New System.Drawing.Font("Arial", 16)
    Dim drawBrush As New _
       System.Drawing.SolidBrush(System.Drawing.Color.Black)
    Dim x As Single = 150.0
    Dim y As Single = 50.0
    Dim drawFormat As New System.Drawing.StringFormat
    formGraphics.DrawString(drawString, drawFont, drawBrush, _
        x, y, drawFormat)
    drawFont.Dispose()
    drawBrush.Dispose()
    formGraphics.Dispose()
End Sub
```

## Compiling the Code

You cannot call the DrawString method in the Load event handler. The drawn content will not be redrawn if the form is resized or obscured by another form. To make your content automatically repaint, you should override the OnPaint method.

## Robust Programming

The following conditions may cause an exception:

- The Arial font is not installed.

## See Also

DrawString
DrawText
FormatFlags
StringFormatFlags

TextFormatFlags
OnPaint
Getting Started with Graphics Programming
How to: Draw Text with GDI

# How to: Draw Vertical Text on a Windows Form

**.NET Framework (current version)**

The following code example shows how to draw vertical text on a form by using the DrawString method of Graphics.

## Example

VB

```
Public Sub DrawVerticalString()
    Dim formGraphics As System.Drawing.Graphics = Me.CreateGraphics()
    Dim drawString As String = "Sample Text"
    Dim drawFont As New System.Drawing.Font("Arial", 16)
    Dim drawBrush As New _
        System.Drawing.SolidBrush(System.Drawing.Color.Black)
    Dim x As Single = 150.0
    Dim y As Single = 50.0
    Dim drawFormat As New System.Drawing.StringFormat
    drawFormat.FormatFlags = StringFormatFlags.DirectionVertical
    formGraphics.DrawString(drawString, drawFont, drawBrush, _
    x, y, drawFormat)
    drawFont.Dispose()
    drawBrush.Dispose()
    formGraphics.Dispose()
End Sub
```

## Compiling the Code

You cannot call this method in the Load event handler. The drawn content will not be redrawn if the form is resized or obscured by another form. To make your content automatically repaint, you should override the OnPaint method.

## Robust Programming

The following conditions may cause an exception:

- The Arial font is not installed.

## See Also

DrawString
FormatFlags

StringFormatFlags
OnPaint
Getting Started with Graphics Programming
Using Fonts and Text

# How to: Render Images with GDI+

**.NET Framework (current version)**

You can use GDI+ to render images that exist as files in your applications. You do this by creating a new object of an Image class (such as Bitmap), creating a Graphics object that refers to the drawing surface you want to use, and calling the DrawImage method of the Graphics object. The image will be painted onto the drawing surface represented by the graphics class. You can use the Image Editor to create and edit image files at design time, and render them with GDI+ at run time. For more information, see Image Editor for Icons.

## To render an image with GDI+

1. Create an object representing the image you want to display. This object must be a member of a class that inherits from Image, such as Bitmap or Metafile. An example is shown:

**VB**

```
' Uses the System.Environment.GetFolderPath to get the path to the
' current user's MyPictures folder.
Dim myBitmap as New Bitmap _
   (System.Environment.GetFolderPath _
      (System.Environment.SpecialFolder.MyPictures))
```

2. Create a Graphics object that represents the drawing surface you want to use. For more information, see How to: Create Graphics Objects for Drawing.

**VB**

```
' Creates a Graphics object that represents the drawing surface of
' Button1.
Dim g as Graphics = Button1.CreateGraphics
```

3. Call the DrawImage of your graphics object to render the image. You must specify both the image to be drawn, and the coordinates where it is to be drawn.

**VB**

```
g.DrawImage(myBitmap, 1, 1)
```

## See Also

- Getting Started with Graphics Programming
- How to: Create Graphics Objects for Drawing
- Pens, Lines, and Rectangles in GDI+
- How to: Draw Text on a Windows Form
- Graphics and Drawing in Windows Forms
- Drawing Lines or Closed Figures (Image Editor for Icons)
- Image Editor for Icons

# How to: Create a Shaped Windows Form

**.NET Framework (current version)**

This example gives a form an elliptical shape that resizes with the form.

## Example

**VB**

```
 Protected Overrides Sub OnPaint( _
ByVal e As System.Windows.Forms.PaintEventArgs)
     Dim shape As New System.Drawing.Drawing2D.GraphicsPath
     shape.AddEllipse(0, 0, Me.Width, Me.Height)
     Me.Region = New System.Drawing.Region(shape)
 End Sub
```

## Compiling the Code

This example requires:

- References to the System.Windows.Forms and System.Drawing namespaces.

This example overrides the OnPaint method to change the shape of the form. To use this code, copy the method declaration as well as the drawing code inside the method.

## See Also

OnPaint
Region
System.Drawing
AddEllipse
Region
Getting Started with Graphics Programming

# How to: Copy Pixels for Reducing Flicker in Windows Forms

**.NET Framework (current version)**

When you animate a simple graphic, users can sometimes encounter flicker or other undesirable visual effects. One way to limit this problem is to use a "bitblt" process on the graphic. Bitblt is the "bit-block transfer" of the color data from an origin rectangle of pixels to a destination rectangle of pixels.

With Windows Forms, bitblt is accomplished using the CopyFromScreen method of the Graphics class. In the parameters of the method, you specify the source and destination (as points), the size of the area to be copied, and the graphics object used to draw the new shape.

In the example below, a shape is drawn on the form in its Paint event handler. Then, the CopyFromScreen method is used to duplicate the shape.

**Note**

Setting the form's DoubleBuffered property to **true** will make graphics-based code in the Paint event be double-buffered. While this will not have any discernable performance gains when using the code below, it is something to keep in mind when working with more complex graphics-manipulation code.

## Example

VB

```
Private Sub Form1_Paint(ByVal sender As Object, ByVal e As _
    System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a circle with a bar on top.
        e.Graphics.FillEllipse(Brushes.DarkBlue, New Rectangle _
             (10, 10, 60, 60))
        e.Graphics.FillRectangle(Brushes.Khaki, New Rectangle _
             (20, 30, 60, 10))
    ' Copy the graphic to a new location.
        e.Graphics.CopyFromScreen(New Point(10, 10), New Point _
             (100, 100), New Size(70, 70))
End Sub
```

## Compiling the Code

The code above is run in the form's Paint event handler so that the graphics persist when the form is redrawn. As such, do not call graphics-related methods in the Load event handler, because the drawn content will not be redrawn if the form is resized or obscured by another form.

## See Also

- CopyPixelOperation
- Graphics.FillRectangle
- Control.OnPaint
- Graphics and Drawing in Windows Forms
- Using a Pen to Draw Lines and Shapes

# Using a Pen to Draw Lines and Shapes

**.NET Framework (current version)**

Use GDI+ **Pen** objects to draw line segments, curves, and the outlines of shapes. In this section, *line* refers to any of these, unless specified to mean only a line segment. Set the properties of a pen to control the color, width, alignment, and style of lines drawn with that pen.

## In This Section

How to: Use a Pen to Draw Lines
: Explains how to draw lines.

How to: Use a Pen to Draw Rectangles
: Describes how to draw rectangles.

How to: Set Pen Width and Alignment
: Explains how to change the width and alignment of a **Pen** object.

How to: Draw a Line with Line Caps
: Describes how to add end caps when drawing a line.

How to: Join Lines
: Shows how to join two lines.

How to: Draw a Custom Dashed Line
: Describes how to draw a dashed line.

How to: Draw a Line Filled with a Texture
: Explains how to draw a texture-filled line.

## Reference

Pen
: Describes this class and has links to all its members.

# Using a Brush to Fill Shapes

**.NET Framework (current version)**

A GDI+ Brush object is used to fill the interior of a closed shape. GDI+ defines several fill styles: solid color, hatch pattern, image texture, and color gradient.

## In This Section

How to: Fill a Shape with a Solid Color
: Describes how to use a solid-color brush to fill shapes.

How to: Fill a Shape with a Hatch Pattern
: Shows how to use a hatch brush to fill shapes.

How to: Fill a Shape with an Image Texture
: Explains how to use a texture brush to fill shapes.

How to: Tile a Shape with an Image
: Describes how to tile an image in a shape.

## Reference

System.Drawing.Brush
: Describes this class and contains links to all of its members

System.Drawing.SolidBrush
: Describes this class and contains links to all of its members

System.Drawing.TextureBrush
: Describes this class and contains links to all of its members.

System.Drawing.Drawing2D.HatchBrush
: Describes this class and contains links to all of its members.

System.Drawing.Drawing2D.PathGradientBrush
: Describes this class and contains links to all of its members.

## Related Sections

Using a Gradient Brush to Fill Shapes
: Contains a list of topics that show how to use a gradient brush.

Using a Pen to Draw Lines and Shapes
: Provides a list of topics that demonstrate how to draw outlined shapes.

Using Managed Graphics Classes

Contains a list of topics describing how to use managed graphics classes.

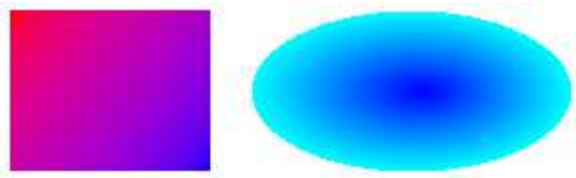
# Using a Gradient Brush to Fill Shapes

**.NET Framework (current version)**

You can use a gradient brush to fill a shape with a gradually changing color. For example, you can use a horizontal gradient to fill a shape with color that changes gradually as you move from the left edge of the shape to the right edge. Imagine a rectangle with a left edge that is black (represented by red, green, and blue components 0, 0, 0) and a right edge that is red (represented by 255, 0, 0). If the rectangle is 256 pixels wide, the red component of a given pixel will be one greater than the red component of the pixel to its left. The leftmost pixel in a row has color components (0, 0, 0), the second pixel has (1, 0, 0), the third pixel has (2, 0, 0), and so on, until you get to the rightmost pixel, which has color components (255, 0, 0). These interpolated color values make up the color gradient.

A linear gradient changes color as you move horizontally, vertically, or parallel to a specified slanted line. A path gradient changes color as you move about the interior and boundary of a path. You can customize path gradients to achieve a wide variety of effects.

The following illustration shows a rectangle filled with a linear gradient brush and an ellipse filled with a path gradient brush.



## In This Section

How to: Create a Linear Gradient
: Shows how to create a linear gradient using the LinearGradientBrush class.

How to: Create a Path Gradient
: Describes how to create a path gradient using the PathGradientBrush class.

How to: Apply Gamma Correction to a Gradient
: Explains how to use gamma correction with a gradient brush.

## Reference

System.Drawing.Drawing2D.LinearGradientBrush
: Contains a description of this class and has links to all of its members.

System.Drawing.Drawing2D.PathGradientBrush
: Contains a description of this class and has links to all of its members.

# Working with Images, Bitmaps, Icons, and Metafiles

**.NET Framework (current version)**

GDI+ provides the **Bitmap** class for working with raster images and the **Metafile** class for working with vector images. The **Bitmap** and the **Metafile** classes both inherit from the **Image** class.

## In This Section

How to: Draw an Existing Bitmap to the Screen
: Describes how to load and draw bitmaps.

How to: Load and Display Metafiles
: Shows how to load and draw metafiles.

Cropping and Scaling Images in GDI+
: Explains how to crop and scale vector and raster images.

How to: Rotate, Reflect, and Skew Images
: Describes how to draw rotated, reflected and skewed images.

How to: Use Interpolation Mode to Control Image Quality During Scaling
: Shows how to use the InterpolationMode enumeration to change image quality.

How to: Create Thumbnail Images
: Describes how to create thumbnail images.

How to: Improve Performance by Avoiding Automatic Scaling
: Explains how to draw an image without automatic scaling.

How to: Read Image Metadata
: Describes how to read metadata from an image.

How to: Create a Bitmap at Run Time
: Shows how to draw a bitmap at runtime.

How to: Extract the Icon Associated with a File in Windows Forms
: Describes how to extract an icon that is an embedded resource of a file.

## Reference

Image
: Describes this class and has links to all of its members.

Metafile

Describes this class and has links to all of its members.

Bitmap

Describes this class and has links to all of its members.

## Related Sections

Images, Bitmaps, and Metafiles

Contains links to topics that discuss different types of bitmaps and manipulating them in your applications.

# Alpha Blending Lines and Fills

**.NET Framework (current version)**

In GDI+, a color is a 32-bit value with 8 bits each for alpha, red, green, and blue. The alpha value indicates the transparency of the color — the extent to which the color is blended with the background color. Alpha values range from 0 through 255, where 0 represents a fully transparent color, and 255 represents a fully opaque color.

Alpha blending is a pixel-by-pixel blending of source and background color data. Each of the three components (red, green, blue) of a given source color is blended with the corresponding component of the background color according to the following formula:

displayColor = sourceColor × alpha / 255 + backgroundColor × (255 – alpha) / 255

For example, suppose the red component of the source color is 150 and the red component of the background color is 100. If the alpha value is 200, the red component of the resultant color is calculated as follows:

150 × 200 / 255 + 100 × (255 – 200) / 255 = 139

## In This Section

How to: Draw Opaque and Semitransparent Lines
: Shows how to draw alpha-blended lines.

How to: Draw with Opaque and Semitransparent Brushes
: Explains how to alpha-blend with brushes.

How to: Use Compositing Mode to Control Alpha Blending
: Describes how to control alpha blending using CompositingMode.

How to: Use a Color Matrix to Set Alpha Values in Images
: Explains how to use a ColorMatrix object to control alpha blending.

# Using Fonts and Text

**.NET Framework (current version)**

There are several classes offered by GDI+ and GDI for drawing text on Windows Forms. The GDI+ Graphics class has several DrawString methods that allow you to specify various features of text, such as location, bounding rectangle, font, and format. In addition, you can draw and measure text with GDI using the static DrawText and MeasureText methods offered by the **TextRenderer** class. The GDI methods also allow you to specify location, font, and format. You can choose either GDI or GDI+ for text rendering; however, GDI generally offers better performance and more accurate text measuring. Other classes that contribute to text rendering include **FontFamily**, **Font**, StringFormat, and **TextFormatFlags**.

## In This Section

How to: Construct Font Families and Fonts
: Shows how to create **Font** and **FontFamily** objects.

How to: Draw Text at a Specified Location
: Describes how to draw text in a certain location using GDI+ and GDI.

How to: Draw Wrapped Text in a Rectangle
: Explains how to draw text in a rectangle using GDI+ and GDI.

How to: Draw Text with GDI
: Demonstrates how to use GDI for drawing text.

How to: Align Drawn Text
: Shows how to format GDI+ and GDI text.

How to: Create Vertical Text
: Describes how to draw vertically aligned text with GDI+.

How to: Set Tab Stops in Drawn Text
: Shows how draw text with tab stops with GDI+.

How to: Enumerate Installed Fonts
: Explains how to list the names of installed fonts.

How to: Create a Private Font Collection
: Describes how to create a PrivateFontCollection object.

How to: Obtain Font Metrics
: Shows how to obtain font metrics such as cell ascent and descent.

How to: Use Antialiasing with Text
: Explains how to use antialiasing when drawing text.

## Reference

Font

Describes this class and contains links to all of its members.

FontFamily

Describes this class and contains links to all of its members.

PrivateFontCollection

Describes this class and contains links to all of its members.

TextRenderer

Describes this class and contains links to all of its members.

TextFormatFlags

Describes this class and contains links to all of its members.

# Constructing and Drawing Curves

**.NET Framework (current version)**

GDI+ supports several types of curves: ellipses, arcs, cardinal splines, and Bézier splines. An ellipse is defined by its bounding rectangle; an arc is a portion of an ellipse defined by a starting angle and a sweep angle. A cardinal spline is defined by an array of points and a tension parameter — the curve passes smoothly through each point in the array, and the tension parameter influences the way the curve bends. A Bézier spline is defined by two endpoints and two control points the curve does not pass through the control points, but the control points influence the direction and bend as the curve goes from one endpoint to the other.

## In This Section

How to: Draw Cardinal Splines
: Describes cardinal splines and how to draw them.

How to: Draw a Single Bézier Spline
: Describes a Bézier spline and how to draw one.

How to: Draw a Sequence of Bézier Splines
: Explains how to draw several Bézier splines in sequence.

# Constructing and Drawing Paths

**.NET Framework (current version)**

A path is a sequence of graphics primitives (lines, rectangles, curves, text, and the like) that can be manipulated and drawn as a single unit. A path can be divided into *figures* that are either open or closed. A figure can contain several primitives.

You can draw a path by calling the DrawPath method of the Graphics class, and you can fill a path by calling the FillPath method of the Graphics class.

## In This Section

How to: Create Figures from Lines, Curves, and Shapes
: Shows how to use a GraphicsPath to create figures.

How to: Fill Open Figures
: Explains how to fill a GraphicsPath.

How to: Flatten a Curved Path into a Line
: Shows how to flatten a GraphicsPath.

## Reference

GraphicsPath
: Describes this class and contains links to all of its members.

# Using Transformations in Managed GDI+

**.NET Framework (current version)**

Affine transformations include rotating, scaling, reflecting, shearing, and translating. In GDI+, the Matrix class provides the foundation for performing affine transformations on vector drawings, images, and text.

## In This Section

Using the World Transformation
: Describes how to scale and rotate graphics using a world transformation matrix.

Why Transformation Order Is Significant
: Demonstrates why the order of transform operations is important.

## Reference

Matrix
: Describes this class and contains links to all of its members.

# Using Graphics Containers

**.NET Framework (current version)**

A Graphics object provides methods such as DrawLine, DrawImage, and DrawString for displaying vector images, raster images, and text. A Graphics object also has several properties that influence the quality and orientation of the items that are drawn. For example, the smoothing mode property determines whether antialiasing is applied to lines and curves, and the world transformation property influences the position and rotation of the items that are drawn.

A Graphics object is associated with a particular display device. When you use a Graphics object to draw in a window, the Graphics object is also associated with that particular window.

A Graphics object can be thought of as a container because it holds a set of properties that influence drawing and it is linked to device-specific information. You can create a secondary container within an existing Graphics object by calling the BeginContainer method of that Graphics object.

## In This Section

Managing the State of a Graphics Object
: Describes how manage the quality settings, clipping area and transformations of a Graphics object.

Using Nested Graphics Containers
: Shows how to use containers to control the state of the Graphics object.

# Using Regions

**.NET Framework (current version)**

The GDI+ Region class allows you to define a custom shape. The shape can be made up of lines, polygons, and curves.

Two common uses for regions are hit testing and clipping. Hit testing is determining whether the mouse was clicked in a certain region of the screen. Clipping is restricting drawing to a certain region.

## In This Section

How to: Use Hit Testing with a Region
: Shows how to use a Region to perform a hit test.

How to: Use Clipping with a Region
: Explains how to set the clipping region for a Graphics object.

## Reference

Region
: Describes this class and contains links to all of its members.

Graphics
: Describes this class and contains links to all of its members.

# Recoloring Images

**.NET Framework (current version)**

Recoloring is the process of adjusting image colors. Some examples of recoloring are changing one color to another, adjusting a color's intensity relative to another color, adjusting the brightness or contrast of all colors, and converting colors to shades of gray.

## In This Section

How to: Use a Color Matrix to Transform a Single Color
: Discusses using a color matrix to transform a color.

How to: Translate Image Colors
: Shows how to translate colors using a color matrix.

Using Transformations to Scale Colors
: Explains how to scale colors using a color matrix.

How to: Rotate Colors
: Describes how to rotate a color using a color matrix.

How to: Shear Colors
: Defines shearing and explains how to shear colors using a color matrix.

How to: Use a Color Remap Table
: Defines remapping and shows how to use a color remap table.

## Reference

ColorMatrix
: Describes this class and contains links to all of its members.

ColorMap
: Describes this class and contains links to all of its members.

## Related Sections

Images, Bitmaps, and Metafiles
: Provides a list of topics regarding the different types of images.

Working with Images, Bitmaps, Icons, and Metafiles
: Contains a list of topics that show how to use different types of images.

Using Managed Graphics Classes
: Contains a list of topics describing how to use managed graphics classes.

# Using Image Encoders and Decoders in Managed GDI+

**.NET Framework (current version)**

The System.Drawing namespace provides the Image and Bitmap classes for storing and manipulating images. By using image encoders in GDI+, you can write images from memory to disk. By using image decoders in GDI+, you can load images from disk into memory. An encoder translates the data in an Image or Bitmap object into a designated disk file format. A decoder translates the data in a disk file to the format required by the Image and Bitmap objects.

GDI+ has built-in encoders and decoders that support the following file types:

- BMP
- GIF
- JPEG
- PNG
- TIFF

GDI+ also has built-in decoders that support the following file types:

- WMF
- EMF
- ICON

The following topics discuss encoders and decoders in more detail:

## In This Section

How to: List Installed Encoders
: Describes how to list the encoders available on a computer.

How to: List Installed Decoders
: Describes how to list the decoders available on a computer.

How to: Determine the Parameters Supported by an Encoder
: Describes how to list the EncoderParameters supported by an encoder.

How to: Convert a BMP image to a PNG image
: Describes how to save a image in a different image format.

How to: Set JPEG Compression Level

Describes how to change the quality level of an image.

## Reference

Image

Bitmap

ImageCodecInfo

EncoderParameter

Encoder

## Related Sections

About GDI+ Managed Code

Images, Bitmaps, and Metafiles

# Using Double Buffering

**.NET Framework (current version)**

You can use double-buffered graphics to reduce flicker in your applications that contain complex painting operations. The .NET Framework contains built-in support for double-buffering or you can manage and render graphics manually.

## In This Section

Double Buffered Graphics
Introduces double buffering concept and outlines .NET Framework support.

How to: Reduce Graphics Flicker with Double Buffering for Forms and Controls
Demonstrates how to use the default double buffering support in the .NET Framework.

How to: Manually Manage Buffered Graphics
Shows how to manage double buffering in applications.

How to: Manually Render Buffered Graphics
Demonstrates how to render double-buffered graphics.

## Reference

SetStyle ,
Control method that enables double buffering.

BufferedGraphicsContext ,
Provides methods for creating graphics buffers.

BufferedGraphicsManager
Provides access to the buffered graphics context for a application domain.