Contents

Enhancing Windows Forms Applications

11/3/2020 • 2 minutes to read • Edit Online

Windows Forms contains many features that you can use to enhance your Windows-based applications to meet the specific needs of your users. The following topics describe these features and how to use them.

In This Section

Graphics and Drawing in Windows Forms
Contains links to topics that describe and show how to use the graphics interface in Windows Forms.

Application Settings for Windows Forms.
Contains links to topics that describe and show how to use the **Application Settings** feature.

Windows Forms Print Support
Contains links to topics that describe and show how to print files from Windows Forms applications.

Drag-and-Drop Operations and Clipboard Support
Contains links to topics that describe and show how to use the drag-and-drop feature and the Clipboard in Windows Forms.

Networking in Windows Forms Applications
Contains links to topics that describe and show how to use networking in Windows Forms.

Globalizing Windows Forms applications
Contains links to topics that show how to globalize Windows Forms applications.

Windows Forms and Unmanaged Applications
Contains links to topics that describe and show how to access COM components from Windows Form applications.

System Information and Windows Forms
Describes how to use system information in Windows Forms.

Power Management in Windows Forms
Describes how to manage power use in Windows Forms applications.

Windows Forms Visual Inheritance
Describes how to inherit from a base form.

Multiple-Document Interface (MDI) Applications
Describes how to create multiple-document interface (MDI) applications.

Integrating User Help in Windows Forms
Describes how to integrate user help in your applications.

Windows Forms Accessibility
Describes how to make your applications available to a wide variety of users.

Using WPF Controls
Describes how to use WPF controls in your Windows Forms-based applications.

Related Sections

Help Systems in Windows Forms Applications

Contains links to topics that describe and show how to provide user help in Windows Forms applications.

Getting Started with Windows Forms

Contains links to topics that describe how to use the basic features of Windows Forms.

Graphics and Drawing in Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

The common language runtime uses an advanced implementation of the Windows Graphics Device Interface (GDI) called GDI+. With GDI+ you can create graphics, draw text, and manipulate graphical images as objects. GDI+ is designed to offer performance and ease of use. You can use GDI+ to render graphical images on Windows Forms and controls. Although you cannot use GDI+ directly on Web Forms, you can display graphical images through the Image Web Server control.

In this section, you will find topics that introduce the fundamentals of GDI+ programming. Although not intended to be a comprehensive reference, this section includes information about the Graphics, Pen, Brush, and Color objects, and explains how to perform such tasks as drawing shapes, drawing text, or displaying images. For more information, see GDI+ Reference.

If you'd like to jump in and get started right away, see Getting Started with Graphics Programming. It has topics on how to use code to draw lines, shapes, text, and more on Windows forms.

In This Section

Graphics Overview
Provides an introduction to the graphics-related managed classes.

About GDI+ Managed Code
Provides information about the managed GDI+ classes.

Using Managed Graphics Classes
Demonstrates how to complete a variety of tasks using the GDI+ managed classes.

Reference

System.Drawing
Provides access to GDI+ basic graphics functionality.

System.Drawing.Drawing2D
Provides advanced two-dimensional and vector graphics functionality.

System.Drawing.Imaging
Provides advanced GDI+ imaging functionality.

System.Drawing.Text
Provides advanced GDI+ typography functionality. The classes in this namespace can be used to create and use collections of fonts.

System.Drawing.Printing
Provides printing functionality.

Related Sections

Custom Control Painting and Rendering
Details how to provide code for painting controls.

Graphics Overview (Windows Forms)

11/3/2020 • 2 minutes to read • Edit Online

GDI+ is a Graphics Device Interface that enables programmers to write device-independent applications. The services of GDI+ are exposed through a set of managed classes.

In This Section

Overview of Graphics
Provides a general introduction to GDI+.

Three Categories of Graphics Services
Describes the three categories that make up programming with GDI+.

Structure of the Graphics Interface
Describes the managed class interface of GDI+.

Reference

System.Drawing
Provides access to GDI+ basic graphics functionality.

System.Drawing.Drawing2D
Provides advanced two-dimensional and vector graphics functionality.

System.Drawing.Imaging
Provides advanced GDI+ imaging functionality.

System.Drawing.Text
Provides advanced GDI+ typography functionality.

System.Drawing.Printing
Provides print-related services.

TextRenderer
Provides GDI text drawing and measuring functionality.

Overview of Graphics

3/9/2021 • 2 minutes to read • Edit Online

GDI+ is an application programming interface (API) that forms the subsystem of the Microsoft Windows operating system. GDI+ is responsible for displaying information on screens and printers. As its name suggests, GDI+ is the successor to GDI, the Graphics Device Interface included with earlier versions of Windows.

Managed Class Interface

The GDI+ API is exposed through a set of classes deployed as managed code. This set of classes is called the *managed class interface* to GDI+. The following namespaces make up the managed class interface:

- System.Drawing
- System.Drawing.Drawing2D
- System.Drawing.Imaging
- System.Drawing.Text
- System.Drawing.Printing

With a Graphics Device Interface, such as GDI+, you can display information on a screen or printer without having to be concerned about the details of a particular display device. The programmer makes calls to methods provided by GDI+ classes. Those methods, in turn, make the appropriate calls to specific device drivers. GDI+ insulates the application from the graphics hardware. It is this insulation that enables a programmer to create device-independent applications.

See also

- Graphics Overview

Three Categories of Graphics Services

11/3/2020 • 2 minutes to read • Edit Online

The graphics offerings in Windows Forms fall into the following three broad categories:

- Two-dimensional (2-D) vector graphics
- Imaging
- Typography

2D Vector Graphics

Two-dimensional vector graphics, such as lines, curves, and figures, are primitives that are specified by sets of points on a coordinate system. For example, a straight line is specified by its two endpoints, and a rectangle is specified by a point giving the location of its upper-left corner and a pair of numbers giving its width and height. A simple path is specified by an array of points that are connected by straight lines. A Bézier spline is a sophisticated curve specified by four control points.

GDI+ provides classes and structures that store information about the primitives themselves, classes that store information about how the primitives will be drawn, and classes that actually do the drawing. For example, the Rectangle structure stores the location and size of a rectangle; the Pen class stores information about line color, line width, and line style; and the Graphics class has methods for drawing lines, rectangles, paths, and other figures. There are also several Brush classes that store information about how closed figures and paths will be filled with colors or patterns.

You can record a vector image, which is a sequence of graphics commands, in a metafile. GDI+ provides the Metafile class for recording, displaying, and saving metafiles. With the MetafileHeader and MetaHeader classes, you can inspect the data stored in a metafile header.

Imaging

Certain kinds of pictures are difficult or impossible to display with the techniques of vector graphics. For example, the pictures on toolbar buttons and the pictures that appear as icons are difficult to specify as collections of lines and curves. A high-resolution digital photograph of a crowded baseball stadium is even more difficult to create with vector techniques. Images of this type are stored as bitmaps, which are arrays of numbers that represent the colors of individual dots on the screen. GDI+ provides the Bitmap class for displaying, manipulating, and saving bitmaps.

Typography

Typography is the display of text in a variety of fonts, sizes, and styles. GDI+ provides extensive support for this complex task. One of the new features in GDI+ is subpixel antialiasing, which gives text rendered on an LCD screen a smoother appearance.

In addition, Windows Forms offers the option to draw text with GDI capabilities in its TextRenderer class.

See also

- Graphics Overview
- About GDI+ Managed Code
- Using Managed Graphics Classes

Structure of the Graphics Interface

11/3/2020 • 2 minutes to read • Edit Online

The managed class interface to GDI+ contains about 60 classes, 50 enumerations, and 8 structures. The Graphics class is at the core of GDI+ functionality; it is the class that actually draws lines, curves, figures, images, and text.

Important Classes

Many classes work together with the Graphics class. For example, the DrawLine method receives a Pen object, which holds attributes (color, width, dash style, and the like) of the line to be drawn. The FillRectangle method can receive a pointer to a LinearGradientBrush object, which works with the Graphics object to fill a rectangle with a gradually changing color. Font and StringFormat objects influence the way a Graphics object draws text. A Matrix object stores and manipulates the world transformation of a Graphics object, which is used to rotate, scale, and flip images.

GDI+ provides several structures (for example, Rectangle, Point, and Size) for organizing graphics data. Also, certain classes serve primarily as structured data types. For example, the BitmapData class is a helper for the Bitmap class, and the PathData class is a helper for the GraphicsPath class.

GDI+ defines several enumerations, which are collections of related constants. For example, the LineJoin enumeration contains the elements Bevel, Miter, and Round, which specify styles that can be used to join two lines.

See also

- Graphics Overview
- About GDI+ Managed Code
- Using Managed Graphics Classes

About GDI+ Managed Code

11/3/2020 • 2 minutes to read • Edit Online

GDI+ is the portion of the Windows operating system that provides two-dimensional vector graphics, imaging, and typography. GDI+ improves on GDI (the Graphics Device Interface included with earlier versions of Windows) by adding new features and by optimizing existing features.

The GDI+ managed class interface (a set of wrappers) is part of the .NET Framework, an environment for building, deploying, and running XML Web services and other applications.

This section provides information about the GDI+ API for programmers using managed code.

In this section

- Lines, Curves, and Shapes
  Discusses vector graphics.
- Images, Bitmaps, and Metafiles
  Discusses the type of images available and how to work with them.
- Coordinate Systems and Transformations
  Discusses how to transform graphics with GDI+.

Reference

- System.Drawing.Graphics
  Describes this class and has links to all its members.
- System.Drawing.Image
  Describes this class and has links to all its members.
- System.Drawing.Bitmap
  Describes this class and has links to all its members.
- System.Drawing.Imaging.Metafile
  Describes this class and has links to all its members.
- System.Drawing.Font
  Describes this class and has links to all its members.
- System.Drawing.Brush
  Describes this class and has links to all its members.
- System.Drawing.Color
  Describes this class and has links to all its members.
- System.Drawing.Drawing2D.Matrix
  Describes this class and has links to all its members.
- System.Windows.Forms.TextRenderer
  Describes this class and has links to all its members.

Related sections

Using Managed Graphics Classes

Contains links to topics that demonstrate how to use the `Graphics` programming interface.

Lines, Curves, and Shapes

11/3/2020 • 2 minutes to read • Edit Online

The vector graphics portion of GDI+ is used to draw lines, draw curves, and to draw and fill shapes.

In This Section

Vector Graphics Overview
Discusses vector graphics.

Pens, Lines, and Rectangles in GDI+
Discusses drawing lines and rectangles.

Ellipses and Arcs in GDI+
Defines arcs and ellipses and identifies the classes needed to draw them.

Polygons in GDI+
Defines polygons and identifies the classes needed to draw them.

Cardinal Splines in GDI+
Defines cardinal splines and identifies the classes needed to draw them.

Bézier Splines in GDI+
Defines Bezier splines and identifies the classes needed to draw them.

Graphics Paths in GDI+
Describes paths and how to create and draw them.

Brushes and Filled Shapes in GDI+
Describes brush types and how to use them.

Open and Closed Curves in GDI+
Defines open and closed curves and how to draw and fill them.

Regions in GDI+
Describes the methods associated with regions.

Restricting the Drawing Surface in GDI+
Describes clipping and how to use it.

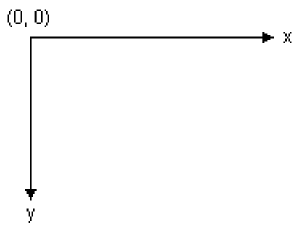
Antialiasing with Lines and Curves
Defines antialiasing and how use antialiasing when drawing lines and curves.

Vector Graphics Overview

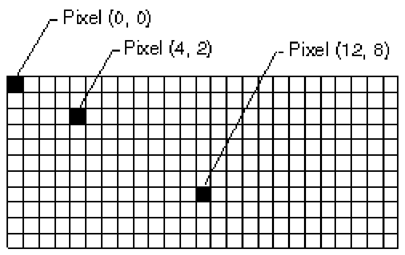
3/9/2021 • 2 minutes to read • Edit Online

GDI+ draws lines, rectangles, and other shapes on a coordinate system. You can choose from a variety of coordinate systems, but the default coordinate system has the origin in the upper-left corner with the x-axis pointing to the right and the y-axis pointing down. The unit of measure in the default coordinate system is the pixel.

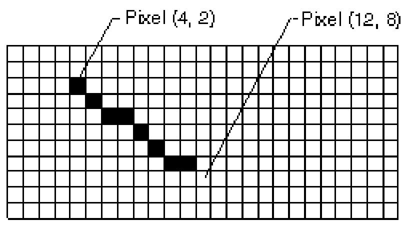
The Building Blocks of GDI+



A computer monitor creates its display on a rectangular array of dots called picture elements or pixels. The number of pixels that appear on the screen varies from one monitor to the next, and the number of pixels that appear on an individual monitor can usually be configured to some extent by the user.



When you use GDI+ to draw a line, rectangle, or curve, you provide certain key information about the item to be drawn. For example, you can specify a line by providing two points, and you can specify a rectangle by providing a point, a height, and a width. GDI+ works in conjunction with the display driver software to determine which pixels must be turned on to show the line, rectangle, or curve. The following illustration shows the pixels that are turned on to display a line from the point (4, 2) to the point (12, 8).



Over time, certain basic building blocks have proven to be the most useful for creating two-dimensional pictures. These building blocks, which are all supported by GDI+, are given in the following list:

- Lines
- Rectangles
- Ellipses
- Arcs

- Polygons
- Cardinal splines
- Bezier splines

Methods For Drawing with a Graphics Object

The Graphics class in GDI+ provides the following methods for drawing the items in the previous list: DrawLine, DrawRectangle, DrawEllipse, DrawPolygon, DrawArc, DrawCurve (for cardinal splines), and DrawBezier. Each of these methods is overloaded; that is, each method supports several different parameter lists. For example, one variation of the DrawLine method receives a Pen object and four integers, while another variation of the DrawLine method receives a Pen object and two Point objects.

The methods for drawing lines, rectangles, and Bézier splines have plural companion methods that draw several items in a single call: DrawLines, DrawRectangles, and DrawBeziers. Also, the DrawCurve method has a companion method, DrawClosedCurve, that closes a curve by connecting the ending point of the curve to the starting point.

All of the drawing methods of the Graphics class work in conjunction with a Pen object. To draw anything, you must create at least two objects: a Graphics object and a Pen object. The Pen object stores attributes, such as line width and color, of the item to be drawn. The Pen object is passed as one of the arguments to the drawing method. For example, one variation of the DrawLine method receives a Pen object and four integers as shown in the following example, which draws a rectangle with a width of 100, a height of 50 and an upper-left corner of (20, 10):

```
myGraphics.DrawRectangle(myPen, 20, 10, 100, 50);
```

```
myGraphics.DrawRectangle(myPen, 20, 10, 100, 50)
```

See also

- System.Drawing.Graphics
- System.Drawing.Pen
- Lines, Curves, and Shapes
- How to: Create Graphics Objects for Drawing

Pens, Lines, and Rectangles in GDI+

11/3/2020 • 2 minutes to read • Edit Online

To draw lines with GDI+ you need to create a Graphics object and a Pen object. The Graphics object provides the methods that actually do the drawing, and the Pen object stores attributes, such as line color, width, and style.

Drawing a Line

To draw a line, call the DrawLine method of the Graphics object. The Pen object is passed as one of the arguments to the DrawLine method. The following example draws a line from the point (4, 2) to the point (12, 6):

```
myGraphics.DrawLine(myPen, 4, 2, 12, 6);
```

```
myGraphics.DrawLine(myPen, 4, 2, 12, 6)
```

DrawLine is an overloaded method of the Graphics class, so there are several ways you can supply it with arguments. For example, you can construct two Point objects and pass the Point objects as arguments to the DrawLine method:

```
Point myStartPoint = new Point(4, 2);
Point myEndPoint = new Point(12, 6);
myGraphics.DrawLine(myPen, myStartPoint, myEndPoint);
```

```
Dim myStartPoint As New Point(4, 2)
Dim myEndPoint As New Point(12, 6)
myGraphics.DrawLine(myPen, myStartPoint, myEndPoint)
```

Constructing a Pen

You can specify certain attributes when you construct a Pen object. For example, one `Pen` constructor allows you to specify color and width. The following example draws a blue line of width 2 from (0, 0) to (60, 30):

```
Pen myPen = new Pen(Color.Blue, 2);
myGraphics.DrawLine(myPen, 0, 0, 60, 30);
```

```
Dim myPen As New Pen(Color.Blue, 2)
myGraphics.DrawLine(myPen, 0, 0, 60, 30)
```

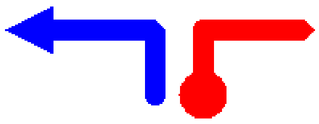
Dashed Lines and Line Caps

The Pen object also exposes properties, such as DashStyle, that you can use to specify features of the line. The following example draws a dashed line from (100, 50) to (300, 80):

```
myPen.DashStyle = DashStyle.Dash;
myGraphics.DrawLine(myPen, 100, 50, 300, 80);
```

```
myPen.DashStyle = DashStyle.Dash
myGraphics.DrawLine(myPen, 100, 50, 300, 80)
```

You can use the properties of the Pen object to set many more attributes of the line. The StartCap and EndCap properties specify the appearance of the ends of the line; the ends can be flat, square, rounded, triangular, or a custom shape. The LineJoin property lets you specify whether connected lines are mitered (joined with sharp corners), beveled, rounded, or clipped. The following illustration shows lines with various cap and join styles.



Drawing a Rectangle

Drawing rectangles with GDI+ is similar to drawing lines. To draw a rectangle, you need a Graphics object and a Pen object. The Graphics object provides a DrawRectangle method, and the Pen object stores attributes, such as line width and color. The Pen object is passed as one of the arguments to the DrawRectangle method. The following example draws a rectangle with its upper-left corner at (100, 50), a width of 80, and a height of 40:

```
myGraphics.DrawRectangle(myPen, 100, 50, 80, 40);
```

```
myGraphics.DrawRectangle(myPen, 100, 50, 80, 40)
```

DrawRectangle is an overloaded method of the Graphics class, so there are several ways you can supply it with arguments. For example, you can construct a Rectangle object and pass the Rectangle object to the DrawRectangle method as an argument:

```
Rectangle myRectangle = new Rectangle(100, 50, 80, 40);
myGraphics.DrawRectangle(myPen, myRectangle);
```

```
Dim myRectangle As New Rectangle(100, 50, 80, 40)
myGraphics.DrawRectangle(myPen, myRectangle)
```

A Rectangle object has methods and properties for manipulating and gathering information about the rectangle. For example, the Inflate and Offset methods change the size and position of the rectangle. The IntersectsWith method tells you whether the rectangle intersects another given rectangle, and the Contains method tells you whether a given point is inside the rectangle.

See also

- System.Drawing.Graphics
- System.Drawing.Pen
- System.Drawing.Rectangle

- How to: Create a Pen
- How to: Draw a Line on a Windows Form
- How to: Draw an Outlined Shape

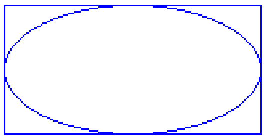
Ellipses and Arcs in GDI+

11/3/2020 • 2 minutes to read • Edit Online

You can easily draw ellipses and arcs using the DrawEllipse and DrawArc methods of the Graphics class.

Drawing an Ellipse

To draw an ellipse, you need a Graphics object and a Pen object. The Graphics object provides the DrawEllipse method, and the Pen object stores attributes, such as width and color, of the line used to render the ellipse. The Pen object is passed as one of the arguments to the DrawEllipse method. The remaining arguments passed to the DrawEllipse method specify the bounding rectangle for the ellipse. The following illustration shows an ellipse along with its bounding rectangle.



The following example draws an ellipse; the bounding rectangle has a width of 80, a height of 40, and an upper-left corner of (100, 50):

```
myGraphics.DrawEllipse(myPen, 100, 50, 80, 40);
```

```
myGraphics.DrawEllipse(myPen, 100, 50, 80, 40)
```

DrawEllipse is an overloaded method of the Graphics class, so there are several ways you can supply it with arguments. For example, you can construct a Rectangle and pass the Rectangle to the DrawEllipse method as an argument:

```
Rectangle myRectangle = new Rectangle(100, 50, 80, 40);
myGraphics.DrawEllipse(myPen, myRectangle);
```

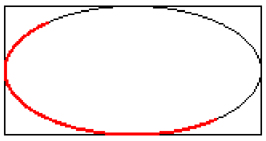
```
Dim myRectangle As New Rectangle(100, 50, 80, 40)
myGraphics.DrawEllipse(myPen, myRectangle)
```

Drawing an Arc

An arc is a portion of an ellipse. To draw an arc, you call the DrawArc method of the Graphics class. The parameters of the DrawArc method are the same as the parameters of the DrawEllipse method, except that DrawArc requires a starting angle and sweep angle. The following example draws an arc with a starting angle of 30 degrees and a sweep angle of 180 degrees:

```
myGraphics.DrawArc(myPen, 100, 50, 140, 70, 30, 180);
myGraphics.DrawArc(myPen, 100, 50, 140, 70, 30, 180)
```

The following illustration shows the arc, the ellipse, and the bounding rectangle.



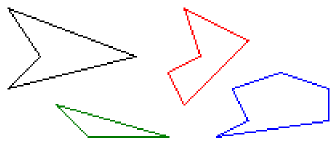
See also

- System.Drawing.Graphics
- System.Drawing.Pen
- Lines, Curves, and Shapes
- How to: Create Graphics Objects for Drawing
- How to: Create a Pen
- How to: Draw an Outlined Shape

Polygons in GDI+

11/3/2020 • 2 minutes to read • Edit Online

A polygon is a closed shape with three or more straight sides. For example, a triangle is a polygon with three sides, a rectangle is a polygon with four sides, and a pentagon is a polygon with five sides. The following illustration shows several polygons.



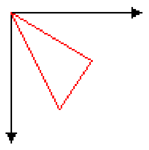
Drawing a Polygon

To draw a polygon, you need a Graphics object, a Pen object, and an array of Point (or PointF) objects. The Graphics object provides the DrawPolygon method. The Pen object stores attributes, such as width and color, of the line used to render the polygon, and the array of Point objects stores the points to be connected by straight lines. The Pen object and the array of Point objects are passed as arguments to the DrawPolygon method. The following example draws a three-sided polygon. Note that there are only three points in `myPointArray`: (0, 0), (50, 30), and (30, 60). The DrawPolygon method automatically closes the polygon by drawing a line from (30, 60) back to the starting point (0, 0).

```
Point[] myPointArray =
{
    new Point(0, 0),
    new Point(50, 30),
    new Point(30, 60)
};
myGraphics.DrawPolygon(myPen, myPointArray);
```

```
Dim myPointArray As Point() = _
   {New Point(0, 0), New Point(50, 30), New Point(30, 60)}
myGraphics.DrawPolygon(myPen, myPointArray)
```

The following illustration shows the polygon.



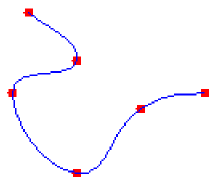
See also

- System.Drawing.Graphics
- System.Drawing.Pen
- Lines, Curves, and Shapes
- How to: Create a Pen

Cardinal Splines in GDI+

11/3/2020 • 2 minutes to read • Edit Online

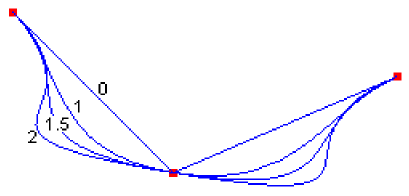
A cardinal spline is a sequence of individual curves joined to form a larger curve. The spline is specified by an array of points and a tension parameter. A cardinal spline passes smoothly through each point in the array; there are no sharp corners and no abrupt changes in the tightness of the curve. The following illustration shows a set of points and a cardinal spline that passes through each point in the set.



Physical and Mathematical Splines

A physical spline is a thin piece of wood or other flexible material. Before the advent of mathematical splines, designers used physical splines to draw curves. A designer would place the spline on a piece of paper and anchor it to a given set of points. The designer could then create a curve by drawing along the spline with a pen or pencil. A given set of points could yield a variety of curves, depending on the properties of the physical spline. For example, a spline with a high resistance to bending would produce a different curve than an extremely flexible spline.

The formulas for mathematical splines are based on the properties of flexible rods, so the curves produced by mathematical splines are similar to the curves that were once produced by physical splines. Just as physical splines of different tension will produce different curves through a given set of points, mathematical splines with different values for the tension parameter will produce different curves through a given set of points. The following illustration shows four cardinal splines passing through the same set of points. The tension is shown for each spline. A tension of 0 corresponds to infinite physical tension, forcing the curve to take the shortest way (straight lines) between points. A tension of 1 corresponds to no physical tension, allowing the spline to take the path of least total bend. With tension values greater than 1, the curve behaves like a compressed spring, pushed to take a longer path.



The four splines in the preceding illustration share the same tangent line at the starting point. The tangent is the line drawn from the starting point to the next point along the curve. Likewise, the shared tangent at the ending point is the line drawn from the ending point to the previous point on the curve.

To draw a cardinal spline, you need an instance of the Graphics class, a Pen, and an array of Point objects The instance of the Graphics class provides the DrawCurve method, which draws the spline, and the Pen stores attributes of the spline, such as line width and color. The array of Point objects stores the points that the curve will pass through. The following code example shows how to draw a cardinal spline that passes through the points in `myPointArray`. The third parameter is the tension.

```
myGraphics.DrawCurve(myPen, myPointArray, 1.5f);
```

```
myGraphics.DrawCurve(myPen, myPointArray, 1.5F)
```

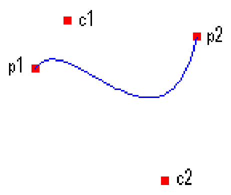
See also

- Lines, Curves, and Shapes
- Constructing and Drawing Curves

Bézier Splines in GDI+

11/3/2020 • 2 minutes to read • Edit Online

A Bézier spline is a curve specified by four points: two end points (p1 and p2) and two control points (c1 and c2). The curve begins at p1 and ends at p2. The curve does not pass through the control points, but the control points act as magnets, pulling the curve in certain directions and influencing the way the curve bends. The following illustration shows a Bézier curve along with its endpoints and control points.



The curve starts at p1 and moves toward the control point c1. The tangent line to the curve at p1 is the line drawn from p1 to c1. The tangent line at the endpoint p2 is the line drawn from c2 to p2.

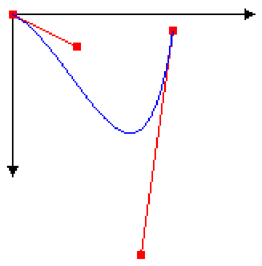
Drawing Bézier Splines

To draw a Bézier spline, you need an instance of the Graphics class and a Pen. The instance of the Graphics class provides the DrawBezier method, and the Pen stores attributes, such as width and color, of the line used to render the curve. The Pen is passed as one of the arguments to the DrawBezier method. The remaining arguments passed to the DrawBezier method are the endpoints and the control points. The following example draws a Bézier spline with starting point (0, 0), control points (40, 20) and (80, 150), and ending point (100, 10):

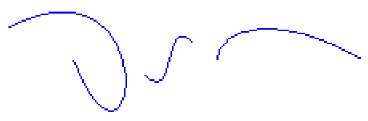
```
myGraphics.DrawBezier(myPen, 0, 0, 40, 20, 80, 150, 100, 10);
```

```
myGraphics.DrawBezier(myPen, 0, 0, 40, 20, 80, 150, 100, 10)
```

The following illustration shows the curve, the control points, and two tangent lines.



Bézier splines were originally developed by Pierre Bézier for design in the automotive industry. They have since proven to be useful in many types of computer-aided design and are also used to define the outlines of fonts. Bézier splines can yield a wide variety of shapes, some of which are shown in the following illustration.



See also

- System.Drawing.Graphics
- System.Drawing.Pen
- Lines, Curves, and Shapes
- Constructing and Drawing Curves
- How to: Create Graphics Objects for Drawing
- How to: Create a Pen

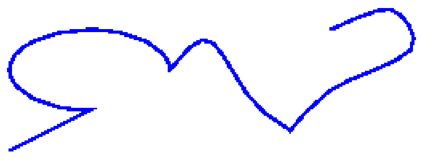
Graphics Paths in GDI+

11/3/2020 • 3 minutes to read • Edit Online

Paths are formed by combining lines, rectangles, and simple curves. Recall from the Vector Graphics Overview that the following basic building blocks have proven to be the most useful for drawing pictures:

- Lines
- Rectangles
- Ellipses
- Arcs
- Polygons
- Cardinal splines
- Bézier splines

In GDI+, the GraphicsPath object allows you to collect a sequence of these building blocks into a single unit. The entire sequence of lines, rectangles, polygons, and curves can then be drawn with one call to the DrawPath method of the Graphics class. The following illustration shows a path created by combining a line, an arc, a Bézier spline, and a cardinal spline.



Using a Path

The GraphicsPath class provides the following methods for creating a sequence of items to be drawn: AddLine, AddRectangle, AddEllipse, AddArc, AddPolygon, AddCurve (for cardinal splines), and AddBezier. Each of these methods is overloaded; that is, each method supports several different parameter lists. For example, one variation of the AddLine method receives four integers, and another variation of the AddLine method receives two Point objects.

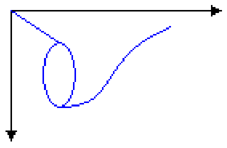
The methods for adding lines, rectangles, and Bézier splines to a path have plural companion methods that add several items to the path in a single call: AddLines, AddRectangles, and AddBeziers. Also, the AddCurve and AddArc methods have companion methods, AddClosedCurve and AddPie, that add a closed curve or pie to the path.

To draw a path, you need a Graphics object, a Pen object, and a GraphicsPath object. The Graphics object provides the DrawPath method, and the Pen object stores attributes, such as width and color, of the line used to render the path. The GraphicsPath object stores the sequence of lines and curves that make up the path. The Pen object and the GraphicsPath object are passed as arguments to the DrawPath method. The following example draws a path that consists of a line, an ellipse, and a Bézier spline:

```
myGraphicsPath.AddLine(0, 0, 30, 20);
myGraphicsPath.AddEllipse(20, 20, 20, 40);
myGraphicsPath.AddBezier(30, 60, 70, 60, 50, 30, 100, 10);
myGraphics.DrawPath(myPen, myGraphicsPath);
```

```
myGraphicsPath.AddLine(0, 0, 30, 20)
myGraphicsPath.AddEllipse(20, 20, 20, 40)
myGraphicsPath.AddBezier(30, 60, 70, 60, 50, 30, 100, 10)
myGraphics.DrawPath(myPen, myGraphicsPath)
```

The following illustration shows the path.



In addition to adding lines, rectangles, and curves to a path, you can add paths to a path. This allows you to combine existing paths to form large, complex paths.

```
myGraphicsPath.AddPath(graphicsPath1, false);
myGraphicsPath.AddPath(graphicsPath2, false);
```

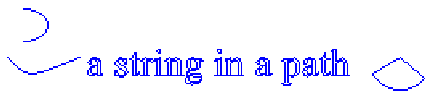
```
myGraphicsPath.AddPath(graphicsPath1, False)
myGraphicsPath.AddPath(graphicsPath2, False)
```

There are two other items you can add to a path: strings and pies. A pie is a portion of the interior of an ellipse. The following example creates a path from an arc, a cardinal spline, a string, and a pie:

```
GraphicsPath myGraphicsPath = new GraphicsPath();

Point[] myPointArray =
{
    new Point(5, 30),
    new Point(20, 40),
    new Point(50, 30)
};

FontFamily myFontFamily = new FontFamily("Times New Roman");
PointF myPointF = new PointF(50, 20);
StringFormat myStringFormat = new StringFormat();

myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180);
myGraphicsPath.StartFigure();
myGraphicsPath.AddCurve(myPointArray);
myGraphicsPath.AddString("a string in a path", myFontFamily,
   0, 24, myPointF, myStringFormat);
myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110);
myGraphics.DrawPath(myPen, myGraphicsPath);
```

```
Dim myGraphicsPath As New GraphicsPath()

Dim myPointArray As Point() = { _
   New Point(5, 30), _
   New Point(20, 40), _
   New Point(50, 30)}

Dim myFontFamily As New FontFamily("Times New Roman")
Dim myPointF As New PointF(50, 20)
Dim myStringFormat As New StringFormat()

myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180)
myGraphicsPath.StartFigure()
myGraphicsPath.AddCurve(myPointArray)
myGraphicsPath.AddString("a string in a path", myFontFamily, _
   0, 24, myPointF, myStringFormat)
myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110)
myGraphics.DrawPath(myPen, myGraphicsPath)
```

The following illustration shows the path. Note that a path does not have to be connected; the arc, cardinal spline, string, and pie are separated.



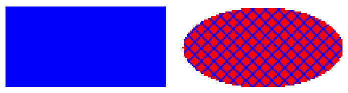
See also

- System.Drawing.Drawing2D.GraphicsPath
- System.Drawing.Point
- Lines, Curves, and Shapes
- How to: Create Graphics Objects for Drawing
- Constructing and Drawing Paths

Brushes and Filled Shapes in GDI+

11/3/2020 • 2 minutes to read • Edit Online

A closed shape, such as a rectangle or an ellipse, consists of an outline and an interior. The outline is drawn with a pen and the interior is filled with a brush. GDI+ provides several brush classes for filling the interiors of closed shapes: SolidBrush, HatchBrush, TextureBrush, LinearGradientBrush, and PathGradientBrush. All of these classes inherit from the Brush class. The following illustration shows a rectangle filled with a solid brush and an ellipse filled with a hatch brush.



Solid Brushes

To fill a closed shape, you need an instance of the Graphics class and a Brush. The instance of the Graphics class provides methods, such as FillRectangle and FillEllipse, and the Brush stores attributes of the fill, such as color and pattern. The Brush is passed as one of the arguments to the fill method. The following code example shows how to fill an ellipse with a solid red color.

```
SolidBrush mySolidBrush = new SolidBrush(Color.Red);
myGraphics.FillEllipse(mySolidBrush, 0, 0, 60, 40);
```

```
Dim mySolidBrush As New SolidBrush(Color.Red)
myGraphics.FillEllipse(mySolidBrush, 0, 0, 60, 40)
```

NOTE
In the preceding example, the brush is of type SolidBrush, which inherits from Brush.

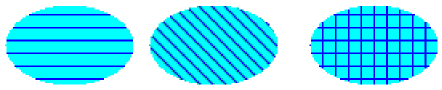
Hatch Brushes

When you fill a shape with a hatch brush, you specify a foreground color, a background color, and a hatch style. The foreground color is the color of the hatching.

```
HatchBrush myHatchBrush =
   new HatchBrush(HatchStyle.Vertical, Color.Blue, Color.Green);
```

```
Dim myHatchBrush As _
   New HatchBrush(HatchStyle.Vertical, Color.Blue, Color.Green)
```

GDI+ provides more than 50 hatch styles; the three styles shown in the following illustration are Horizontal, ForwardDiagonal, and Cross.



Texture Brushes

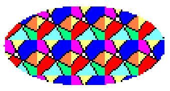
With a texture brush, you can fill a shape with a pattern stored in a bitmap. For example, suppose the following picture is stored in a disk file named `MyTexture.bmp`.

The following code example shows how to fill an ellipse by repeating the picture stored in `MyTexture.bmp`.

```
Image myImage = Image.FromFile("MyTexture.bmp");
TextureBrush myTextureBrush = new TextureBrush(myImage);
myGraphics.FillEllipse(myTextureBrush, 0, 0, 100, 50);
```

```
Dim myImage As Image = Image.FromFile("MyTexture.bmp")
Dim myTextureBrush As New TextureBrush(myImage)
myGraphics.FillEllipse(myTextureBrush, 0, 0, 100, 50)
```

The following illustration shows the filled ellipse.



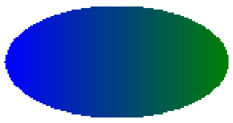
Gradient Brushes

GDI+ provides two kinds of gradient brushes: linear and path. You can use a linear gradient brush to fill a shape with color that changes gradually as you move across the shape horizontally, vertically, or diagonally. The following code example shows how to fill an ellipse with a horizontal gradient brush that changes from blue to green as you move from the left edge of the ellipse to the right edge.

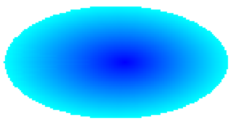
```
LinearGradientBrush myLinearGradientBrush = new LinearGradientBrush(
   myRectangle,
   Color.Blue,
   Color.Green,
   LinearGradientMode.Horizontal);
myGraphics.FillEllipse(myLinearGradientBrush, myRectangle);
```

```
Dim myLinearGradientBrush As New LinearGradientBrush( _
   myRectangle, _
   Color.Blue, _
   Color.Green, _
   LinearGradientMode.Horizontal)
myGraphics.FillEllipse(myLinearGradientBrush, myRectangle)
```

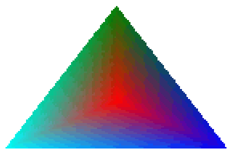
The following illustration shows the filled ellipse.



A path gradient brush can be configured to change color as you move from the center of a shape toward the edge.



Path gradient brushes are quite flexible. The gradient brush used to fill the triangle in the following illustration changes gradually from red at the center to each of three different colors at the vertices.



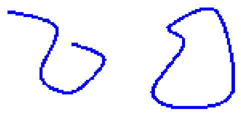
See also

- System.Drawing.SolidBrush
- System.Drawing.Drawing2D.HatchBrush
- System.Drawing.TextureBrush
- System.Drawing.Drawing2D.LinearGradientBrush
- Lines, Curves, and Shapes
- How to: Draw a Filled Rectangle on a Windows Form
- How to: Draw a Filled Ellipse on a Windows Form

Open and Closed Curves in GDI+

11/3/2020 • 2 minutes to read • Edit Online

The following illustration shows two curves: one open and one closed.



Managed Interface for Curves

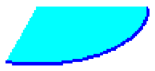
Closed curves have an interior and therefore can be filled with a brush. The Graphics class in GDI+ provides the following methods for filling closed shapes and curves: FillRectangle, FillEllipse, FillPie, FillPolygon, FillClosedCurve, FillPath, and FillRegion. Whenever you call one of these methods, you must pass one of the specific brush types (SolidBrush, HatchBrush, TextureBrush, LinearGradientBrush, or PathGradientBrush) as an argument.

The FillPie method is a companion to the DrawArc method. Just as the DrawArc method draws a portion of the outline of an ellipse, the FillPie method fills a portion of the interior of an ellipse. The following example draws an arc and fills the corresponding portion of the interior of the ellipse:

```
myGraphics.FillPie(mySolidBrush, 0, 0, 140, 70, 0, 120);
myGraphics.DrawArc(myPen, 0, 0, 140, 70, 0, 120);
```

```
myGraphics.FillPie(mySolidBrush, 0, 0, 140, 70, 0, 120)
myGraphics.DrawArc(myPen, 0, 0, 140, 70, 0, 120)
```

The following illustration shows the arc and the filled pie.



The FillClosedCurve method is a companion to the DrawClosedCurve method. Both methods automatically close the curve by connecting the ending point to the starting point. The following example draws a curve that passes through (0, 0), (60, 20), and (40, 50). Then, the curve is automatically closed by connecting (40, 50) to the starting point (0, 0), and the interior is filled with a solid color.

```
Point[] myPointArray =
{
    new Point(0, 0),
    new Point(60, 20),
    new Point(40, 50)
};
myGraphics.DrawClosedCurve(myPen, myPointArray);
myGraphics.FillClosedCurve(mySolidBrush, myPointArray);
```

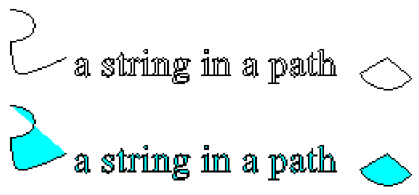
```
Dim myPointArray As Point() = _
   {New Point(0, 0), New Point(60, 20), New Point(40, 50)}
myGraphics.DrawClosedCurve(myPen, myPointArray)
myGraphics.FillClosedCurve(mySolidBrush, myPointArray)
```

The FillPath method fills the interiors of the separate pieces of a path. If a piece of a path doesn't form a closed curve or shape, the FillPath method automatically closes that piece of the path before filling it. The following example draws and fills a path that consists of an arc, a cardinal spline, a string, and a pie:

```
SolidBrush mySolidBrush = new SolidBrush(Color.Aqua);
GraphicsPath myGraphicsPath = new GraphicsPath();

Point[] myPointArray =
{
    new Point(15, 20),
    new Point(20, 40),
    new Point(50, 30)
};

FontFamily myFontFamily = new FontFamily("Times New Roman");
PointF myPointF = new PointF(50, 20);
StringFormat myStringFormat = new StringFormat();

myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180);
myGraphicsPath.AddCurve(myPointArray);
myGraphicsPath.AddString("a string in a path", myFontFamily,
   0, 24, myPointF, myStringFormat);
myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110);

myGraphics.FillPath(mySolidBrush, myGraphicsPath);
myGraphics.DrawPath(myPen, myGraphicsPath);
```

```
Dim mySolidBrush As New SolidBrush(Color.Aqua)
Dim myGraphicsPath As New GraphicsPath()

Dim myPointArray As Point() = { _
   New Point(15, 20), _
   New Point(20, 40), _
   New Point(50, 30)}

Dim myFontFamily As New FontFamily("Times New Roman")
Dim myPointF As New PointF(50, 20)
Dim myStringFormat As New StringFormat()

myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180)
myGraphicsPath.AddCurve(myPointArray)
myGraphicsPath.AddString("a string in a path", myFontFamily, _
   0, 24, myPointF, myStringFormat)
myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110)

myGraphics.FillPath(mySolidBrush, myGraphicsPath)
myGraphics.DrawPath(myPen, myGraphicsPath)
```

The following illustration shows the path with and without the solid fill. Note that the text in the string is outlined, but not filled, by the DrawPath method. It is the FillPath method that paints the interiors of the characters in the string.



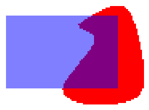
See also

- System.Drawing.Drawing2D.GraphicsPath
- System.Drawing.Pen
- System.Drawing.Point
- Lines, Curves, and Shapes
- How to: Create Graphics Objects for Drawing
- Constructing and Drawing Paths

Regions in GDI+

11/3/2020 • 2 minutes to read • Edit Online

A region is a portion of the display area of an output device. Regions can be simple (a single rectangle) or complex (a combination of polygons and closed curves). The following illustration shows two regions: one constructed from a rectangle, and the other constructed from a path.

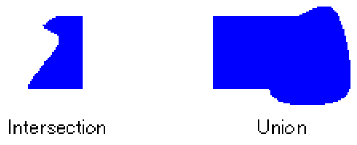


Using Regions

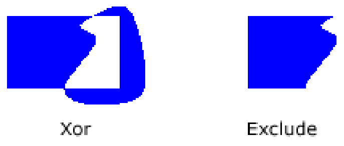
Regions are often used for clipping and hit testing. Clipping involves restricting drawing to a certain region of the display area, usually the portion that needs to be updated. Hit testing involves checking to determine whether the cursor is in a certain region of the screen when a mouse button is pressed.

You can construct a region from a rectangle or a path. You can also create complex regions by combining existing regions. The Region class provides the following methods for combining regions: Intersect, Union, Xor, Exclude, and Complement.

The intersection of two regions is the set of all points belonging to both regions. The union is the set of all points belonging to one or the other or both regions. The complement of a region is the set of all points that are not in the region. The following illustration shows the intersection and union of two regions shown in the preceding illustration.



The Xor method, applied to a pair of regions, produces a region that contains all points that belong to one region or the other, but not both. The Exclude method, applied to a pair of regions, produces a region that contains all points in the first region that are not in the second region. The following illustration shows the regions that result from applying the Xor and Exclude methods to the two regions shown at the beginning of this topic.



To fill a region, you need a Graphics object, a Brush object, and a Region object. The Graphics object provides the FillRegion method, and the Brush object stores attributes of the fill, such as color or pattern. The following example fills a region with a solid color.

```
myGraphics.FillRegion(mySolidBrush, myRegion);
```

```
myGraphics.FillRegion(mySolidBrush, myRegion)
```

See also

- System.Drawing.Region
- Lines, Curves, and Shapes
- Using Regions

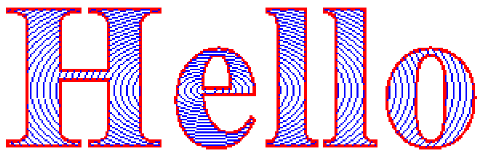
Restricting the Drawing Surface in GDI+

11/3/2020 • 2 minutes to read • Edit Online

Clipping involves restricting drawing to a certain rectangle or region. The following illustration shows the string "Hello" clipped to a heart-shaped region.

Clipping with Regions

Regions can be constructed from paths, and paths can contain the outlines of strings, so you can use outlined text for clipping. The following illustration shows a set of concentric ellipses clipped to the interior of a string of text.



To draw with clipping, create a Graphics object, set its Clip property, and then call the drawing methods of that same Graphics object:

```
myGraphics.Clip = myRegion;
myGraphics.DrawLine(myPen, 0, 0, 200, 200);
```

```
myGraphics.Clip = myRegion
myGraphics.DrawLine(myPen, 0, 0, 200, 200)
```

See also

- System.Drawing.Graphics
- System.Drawing.Region
- Lines, Curves, and Shapes
- Using Regions

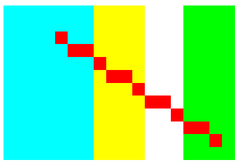
Antialiasing with Lines and Curves

11/3/2020 • 2 minutes to read • Edit Online

When you use GDI+ to draw a line, you provide the starting point and ending point of the line, but you do not have to provide any information about the individual pixels on the line. GDI+ works in conjunction with the display driver software to determine which pixels will be turned on to show the line on a particular display device.

Aliasing

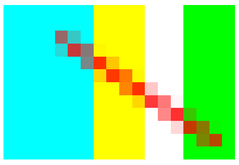
Consider the straight red line that goes from the point (4, 2) to the point (16, 10). Assume the coordinate system has its origin in the upper-left corner and that the unit of measure is the pixel. Also assume that the x-axis points to the right and the y-axis points down. The following illustration shows an enlarged view of the red line drawn on a multicolored background.



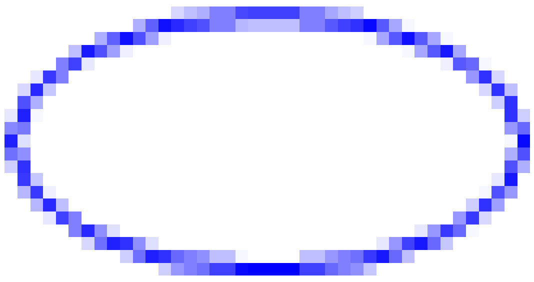
The red pixels used to render the line are opaque. There are no partially transparent pixels in the line. This type of line rendering gives the line a jagged appearance, and the line looks somewhat like a staircase. This technique of representing a line with a staircase is called aliasing; the staircase is an alias for the theoretical line.

Antialiasing

A more sophisticated technique for rendering a line involves using partially transparent pixels along with opaque pixels. Pixels are set to pure red, or to some blend of red and the background color, depending on how close they are to the line. This type of rendering is called antialiasing and results in a line that the human eye perceives as more smooth. The following illustration shows how certain pixels are blended with the background to produce an antialiased line.



Antialiasing, also called smoothing, can also be applied to curves. The following illustration shows an enlarged view of a smoothed ellipse.



The following illustration shows the same ellipse in its actual size, once without antialiasing and once with antialiasing.

Without antialiasing
With antialiasing

To draw lines and curves that use antialiasing, create an instance of the Graphics class and set its SmoothingMode property to AntiAlias or HighQuality. Then call one of the drawing methods of that same Graphics class.

```
myGraphics.SmoothingMode = SmoothingMode.AntiAlias;
myGraphics.DrawLine(myPen, 0, 0, 12, 8);
```

```
myGraphics.SmoothingMode = SmoothingMode.AntiAlias
myGraphics.DrawLine(myPen, 0, 0, 12, 8)
```

See also

- System.Drawing.Drawing2D.SmoothingMode
- Lines, Curves, and Shapes
- How to: Use Antialiasing with Text

Images, Bitmaps, and Metafiles

11/3/2020 • 2 minutes to read • Edit Online

The `Image` class is an abstract base class that provides methods for working with raster images (bitmaps) and vector images (metafiles). The `Bitmap` class and the Metafile class both inherit from the `Image` class. The `Bitmap` class expands on the capabilities of the `Image` class by providing additional methods for loading, saving, and manipulating raster images. The Metafile class expands on the capabilities of the `Image` class by providing additional methods for recording and examining vector images.

In This Section

Types of Bitmaps
Discusses the various image formats.

Metafiles in GDI+
Discusses GDI+ support for metafiles.

Drawing, Positioning, and Cloning Images in GDI+
Discusses methods for drawing vector and raster images with managed code.

Cropping and Scaling Images in GDI+
Discusses methods for cropping and scaling vector and raster images with managed code

Reference

Image
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

Working with Images, Bitmaps, Icons, and Metafiles
Contains links to topics that demonstrate how to use images in your application.

Types of Bitmaps

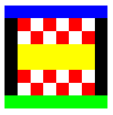
3/9/2021 • 6 minutes to read • Edit Online

A bitmap is an array of bits that specify the color of each pixel in a rectangular array of pixels. The number of bits devoted to an individual pixel determines the number of colors that can be assigned to that pixel. For example, if each pixel is represented by 4 bits, then a given pixel can be assigned one of 16 different colors (2^4 = 16). The following table shows a few examples of the number of colors that can be assigned to a pixel represented by a given number of bits.

BITS PER PIXEL	NUMBER OF COLORS THAT CAN BE ASSIGNED TO A PIXEL
1	2^1 = 2
2	2^2 = 4
4	2^4 = 16
8	2^8 = 256
16	2^16 = 65,536
24	2^24 = 16,777,216

Disk files that store bitmaps usually contain one or more information blocks that store information such as the number of bits per pixel, number of pixels in each row, and number of rows in the array. Such a file might also contain a color table (sometimes called a color palette). A color table maps numbers in the bitmap to specific colors. The following illustration shows an enlarged image along with its bitmap and color table. Each pixel is represented by a 4-bit number, so there are 2^4 = 16 colors in the color table. Each color in the table is represented by a 24-bit number: 8 bits for red, 8 bits for green, and 8 bits for blue. The numbers are shown in hexadecimal (base 16) form: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```



0	000000	⬛
1	FF0000	🟥
2	00FF00	🟩
3	0000FF	🟦
4	FFFFFF	⬜
5	FFFF00	🟨
6	FF00FF	🟪
7	00FFFF	🟦
8	FF0080	🟥
9	FF8040	🟧
A	804000	🟫
B	008080	🟩
C	800000	🟥
D	800080	🟪
E	8080FF	🟦

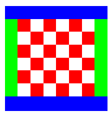
Look at the pixel in row 3, column 5 of the image. The corresponding number in the bitmap is 1. The color table tells us that 1 represents the color red so the pixel is red. All the entries in the top row of the bitmap are 3. The color table tells us that 3 represents blue, so all the pixels in the top row of the image are blue.

NOTE

Some bitmaps are stored in bottom-up format; the numbers in the first row of the bitmap correspond to the pixels in the bottom row of the image.

A bitmap that stores indexes into a color table is called a palette-indexed bitmap. Some bitmaps have no need for a color table. For example, if a bitmap uses 24 bits per pixel, that bitmap can store the colors themselves rather than indexes into a color table. The following illustration shows a bitmap that stores colors directly (24 bits per pixel) rather than using a color table. The illustration also shows an enlarged view of the corresponding image. In the bitmap, FFFFFF represents white, FF0000 represents red, 00FF00 represents green, and 0000FF represents blue.

```
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
```



Graphics File Formats

There are many standard formats for saving bitmaps in disk files. GDI+ supports the graphics file formats described in the following paragraphs.

BMP

BMP is a standard format used by Windows to store device-independent and application-independent images. The number of bits per pixel (1, 4, 8, 15, 24, 32, or 64) for a given BMP file is specified in a file header. BMP files with 24 bits per pixel are common. BMP files are usually not compressed and, therefore, are not well suited for transfer across the Internet.

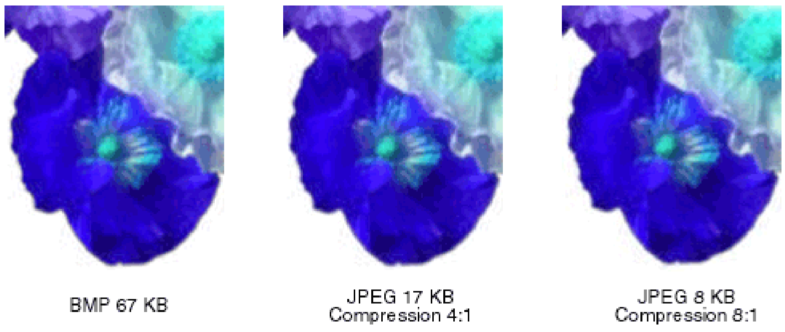
Graphics Interchange Format (GIF)

GIF is a common format for images that appear on Web pages. GIFs work well for line drawings, pictures with blocks of solid color, and pictures with sharp boundaries between colors. GIFs are compressed, but no information is lost in the compression process; a decompressed image is exactly the same as the original. One color in a GIF can be designated as transparent, so that the image will have the background color of any Web page that displays it. A sequence of GIF images can be stored in a single file to form an animated GIF. GIFs store at most 8 bits per pixel, so they are limited to 256 colors.

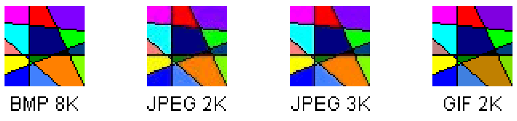
Joint Photographic Experts Group (JPEG)

JPEG is a compression scheme that works well for natural scenes such as scanned photographs. Some information is lost in the compression process, but often the loss is imperceptible to the human eye. JPEGs store 24 bits per pixel, so they are capable of displaying more than 16 million colors. JPEGs do not support transparency or animation.

The level of compression in JPEG images is configurable, but higher compression levels (smaller files) result in more loss of information. A 20:1 compression ratio often produces an image that the human eye finds difficult to distinguish from the original. The following illustration shows a BMP image and two JPEG images that were compressed from that BMP image. The first JPEG has a compression ratio of 4:1 and the second JPEG has a compression ratio of about 8:1.



JPEG compression does not work well for line drawings, blocks of solid color, and sharp boundaries. The following illustration shows a BMP along with two JPEGs and a GIF. The JPEGs and the GIF were compressed from the BMP. The compression ratio is 4:1 for the GIF, 4:1 for the smaller JPEG, and 8:3 for the larger JPEG. Note that the GIF maintains the sharp boundaries along the lines, but the JPEGs tend to blur the boundaries.



JPEG is a compression scheme, not a file format. JPEG File Interchange Format (JFIF) is a file format commonly used for storing and transferring images that have been compressed according to the JPEG scheme. JFIF files displayed by Web browsers use the .jpg extension.

Exchangeable Image File (EXIF)

EXIF is a file format used for photographs captured by digital cameras. An EXIF file contains an image that is compressed according to the JPEG specification. An EXIF file also contains information about the photograph (date taken, shutter speed, exposure time, and so on) and information about the camera (manufacturer, model, and so on).

Portable Network Graphics (PNG)

The PNG format retains many of the advantages of the GIF format but also provides capabilities beyond those of GIF. Like GIF files, PNG files are compressed with no loss of information. PNG files can store colors with 8, 24, or 48 bits per pixel and grayscales with 1, 2, 4, 8, or 16 bits per pixel. In contrast, GIF files can use only 1, 2, 4, or 8 bits per pixel. A PNG file can also store an alpha value for each pixel, which specifies the degree to which the color of that pixel is blended with the background color.

PNG improves on GIF in its ability to progressively display an image (that is, to display better and better approximations of the image as it arrives over a network connection). PNG files can contain gamma correction and color correction information so that the images can be accurately rendered on a variety of display devices.

Tag Image File Format (TIFF)

TIFF is a flexible and extendable format that is supported by a wide variety of platforms and image-processing applications. TIFF files can store images with an arbitrary number of bits per pixel and can employ a variety of compression algorithms. Several images can be stored in a single, multiple-page TIFF file. Information related to the image (scanner make, host computer, type of compression, orientation, samples per pixel, and so on) can be stored in the file and arranged through the use of tags. The TIFF format can be extended as needed by the approval and addition of new tags.

See also

- System.Drawing.Image
- System.Drawing.Bitmap
- System.Drawing.Imaging.PixelFormat
- Images, Bitmaps, and Metafiles

- Working with Images, Bitmaps, Icons, and Metafiles

Metafiles in GDI+

11/3/2020 • 2 minutes to read • Edit Online

GDI+ provides the Metafile class so that you can record and display metafiles. A metafile, also called a vector image, is an image that is stored as a sequence of drawing commands and settings. The commands and settings recorded in a Metafile object can be stored in memory or saved to a file or stream.

Metafile Formats

GDI+ can display metafiles that have been stored in the following formats:

- Windows Metafile (WMF)
- Enhanced Metafile (EMF)
- EMF+

GDI+ can record metafiles in the EMF and EMF+ formats, but not in the WMF format.

EMF+ is an extension to EMF that allows GDI+ records to be stored. There are two variations on the EMF+ format: EMF+ Only and EMF+ Dual. EMF+ Only metafiles contain only GDI+ records. Such metafiles can be displayed by GDI+ but not by GDI. EMF+ Dual metafiles contain GDI+ and GDI records. Each GDI+ record in an EMF+ Dual metafile is paired with an alternate GDI record. Such metafiles can be displayed by GDI+ or by GDI.

The following example displays a metafile that was previously saved as a file. The metafile is displayed with its upper-left corner at (100, 100).

```
public void Example_DisplayMetafile(PaintEventArgs e)
{
    Graphics myGraphics = e.Graphics;
    Metafile myMetafile = new Metafile("SampleMetafile.emf");
    myGraphics.DrawImage(myMetafile, 100, 100);
}
```

```
Public Sub Example_DisplayMetafile(ByVal e As PaintEventArgs)
    Dim myGraphics As Graphics = e.Graphics
    Dim myMetafile As New Metafile("SampleMetafile.emf")
    myGraphics.DrawImage(myMetafile, 100, 100)
End Sub
```

See also

- Images, Bitmaps, and Metafiles

Drawing, Positioning, and Cloning Images in GDI+

11/3/2020 • 2 minutes to read • Edit Online

You can use the Bitmap class to load and display raster images, and you can use the Metafile class to load and display vector images. The Bitmap and Metafile classes inherit from the Image class. To display a vector image, you need an instance of the Graphics class and a Metafile. To display a raster image, you need an instance of the Graphics class and a Bitmap. The instance of the Graphics class provides the DrawImage method, which receives the Metafile or Bitmap as an argument.

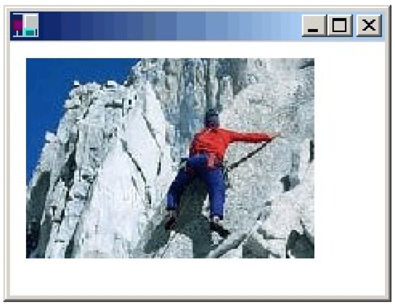
File Types and Cloning

The following code example shows how to construct a Bitmap from the file Climber.jpg and displays the bitmap. The destination point for the upper-left corner of the image, (10, 10), is specified in the second and third parameters.

```
Bitmap myBitmap = new Bitmap("Climber.jpg");
myGraphics.DrawImage(myBitmap, 10, 10);
```

```
Dim myBitmap As New Bitmap("Climber.jpg")
myGraphics.DrawImage(myBitmap, 10, 10)
```

The following illustration shows the image.



You can construct Bitmap objects from a variety of graphics file formats: BMP, GIF, JPEG, EXIF, PNG, TIFF, and ICON.

The following code example shows how to construct Bitmap objects from a variety of file types and then displays the bitmaps.

```
Bitmap myBMP = new Bitmap("SpaceCadet.bmp");
Bitmap myGIF = new Bitmap("Soda.gif");
Bitmap myJPEG = new Bitmap("Mango.jpg");
Bitmap myPNG = new Bitmap("Flowers.png");
Bitmap myTIFF = new Bitmap("MS.tif");

myGraphics.DrawImage(myBMP, 10, 10);
myGraphics.DrawImage(myGIF, 220, 10);
myGraphics.DrawImage(myJPEG, 280, 10);
myGraphics.DrawImage(myPNG, 150, 200);
myGraphics.DrawImage(myTIFF, 300, 200);
```

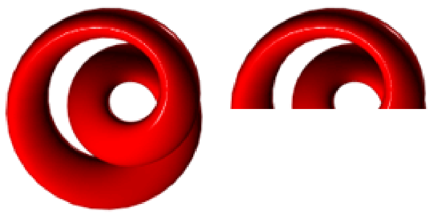
```
Dim myBMP As New Bitmap("SpaceCadet.bmp")
Dim myGIF As New Bitmap("Soda.gif")
Dim myJPEG As New Bitmap("Mango.jpg")
Dim myPNG As New Bitmap("Flowers.png")
Dim myTIFF As New Bitmap("MS.tif")

myGraphics.DrawImage(myBMP, 10, 10)
myGraphics.DrawImage(myGIF, 220, 10)
myGraphics.DrawImage(myJPEG, 280, 10)
myGraphics.DrawImage(myPNG, 150, 200)
myGraphics.DrawImage(myTIFF, 300, 200)
```

The Bitmap class provides a Clone method that you can use to make a copy of an existing Bitmap. The Clone method has a source rectangle parameter that you can use to specify the portion of the original bitmap that you want to copy. The following code example shows how to create a Bitmap by cloning the top half of an existing Bitmap. Then both images are drawn.

```
Bitmap originalBitmap = new Bitmap("Spiral.png");
Rectangle sourceRectangle = new Rectangle(0, 0, originalBitmap.Width,
   originalBitmap.Height / 2);

Bitmap secondBitmap = originalBitmap.Clone(sourceRectangle,
   PixelFormat.DontCare);

myGraphics.DrawImage(originalBitmap, 10, 10);
myGraphics.DrawImage(secondBitmap, 150, 10);
```

```
Dim originalBitmap As New Bitmap("Spiral.png")
Dim sourceRectangle As New Rectangle(0, 0, originalBitmap.Width, _
   CType(originalBitmap.Height / 2, Integer))

Dim secondBitmap As Bitmap = originalBitmap.Clone(sourceRectangle, _
   PixelFormat.DontCare)

myGraphics.DrawImage(originalBitmap, 10, 10)
myGraphics.DrawImage(secondBitmap, 150, 10)
```

The following illustration shows the two images.



See also

- Images, Bitmaps, and Metafiles
- How to: Create Graphics Objects for Drawing
- Working with Images, Bitmaps, Icons, and Metafiles

Cropping and Scaling Images in GDI+

11/3/2020 • 2 minutes to read • Edit Online

You can use the DrawImage method of the Graphics class to draw and position vector images and raster images. DrawImage is an overloaded method, so there are several ways you can supply it with arguments.

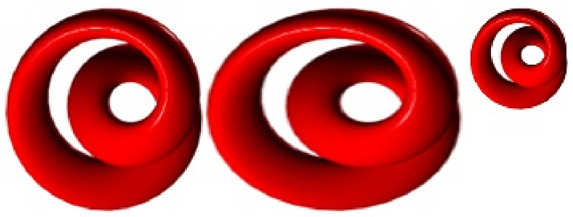
DrawImage Variations

One variation of the DrawImage method receives a Bitmap and a Rectangle. The rectangle specifies the destination for the drawing operation; that is, it specifies the rectangle in which to draw the image. If the size of the destination rectangle is different from the size of the original image, the image is scaled to fit the destination rectangle. The following code example shows how to draw the same image three times: once with no scaling, once with an expansion, and once with a compression:

```
Bitmap myBitmap = new Bitmap("Spiral.png");

Rectangle expansionRectangle = new Rectangle(135, 10,
   myBitmap.Width, myBitmap.Height);

Rectangle compressionRectangle = new Rectangle(300, 10,
   myBitmap.Width / 2, myBitmap.Height / 2);

myGraphics.DrawImage(myBitmap, 10, 10);
myGraphics.DrawImage(myBitmap, expansionRectangle);
myGraphics.DrawImage(myBitmap, compressionRectangle);
```

```
Dim myBitmap As New Bitmap("Spiral.png")

Dim expansionRectangle As New Rectangle(135, 10, _
   myBitmap.Width, myBitmap.Height)

Dim compressionRectangle As New Rectangle(300, 10, _
   CType(myBitmap.Width / 2, Integer), CType(myBitmap.Height / 2, Integer))

myGraphics.DrawImage(myBitmap, 10, 10)
myGraphics.DrawImage(myBitmap, expansionRectangle)
myGraphics.DrawImage(myBitmap, compressionRectangle)
```

The following illustration shows the three pictures.



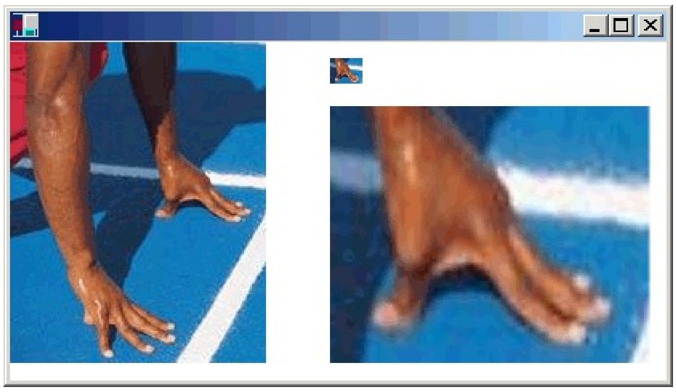
Some variations of the DrawImage method have a source-rectangle parameter as well as a destination-rectangle parameter. The source-rectangle parameter specifies the portion of the original image to draw. The destination rectangle specifies the rectangle in which to draw that portion of the image. If the size of the destination rectangle is different from the size of the source rectangle, the picture is scaled to fit the destination rectangle.

The following code example shows how to construct a Bitmap from the file Runner.jpg. The entire image is drawn with no scaling at (0, 0). Then a small portion of the image is drawn twice: once with a compression and once with an expansion.

```
Bitmap myBitmap = new Bitmap("Runner.jpg");

// One hand of the runner
Rectangle sourceRectangle = new Rectangle(80, 70, 80, 45);

// Compressed hand
Rectangle destRectangle1 = new Rectangle(200, 10, 20, 16);

// Expanded hand
Rectangle destRectangle2 = new Rectangle(200, 40, 200, 160);

// Draw the original image at (0, 0).
myGraphics.DrawImage(myBitmap, 0, 0);

// Draw the compressed hand.
myGraphics.DrawImage(
   myBitmap, destRectangle1, sourceRectangle, GraphicsUnit.Pixel);

// Draw the expanded hand.
myGraphics.DrawImage(
   myBitmap, destRectangle2, sourceRectangle, GraphicsUnit.Pixel);
```

```
Dim myBitmap As New Bitmap("Runner.jpg")

' One hand of the runner
Dim sourceRectangle As New Rectangle(80, 70, 80, 45)

' Compressed hand
Dim destRectangle1 As New Rectangle(200, 10, 20, 16)

' Expanded hand
Dim destRectangle2 As New Rectangle(200, 40, 200, 160)

' Draw the original image at (0, 0).
myGraphics.DrawImage(myBitmap, 0, 0)

' Draw the compressed hand.
myGraphics.DrawImage( _
   myBitmap, destRectangle1, sourceRectangle, GraphicsUnit.Pixel)

' Draw the expanded hand.
myGraphics.DrawImage( _
   myBitmap, destRectangle2, sourceRectangle, GraphicsUnit.Pixel)
```

The following illustration shows the unscaled image, and the compressed and expanded image portions.



See also

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

Coordinate Systems and Transformations

11/3/2020 • 2 minutes to read • Edit Online

GDI+ provides a world transformation and a page transformation so that you can transform (rotate, scale, translate, and so on) the items you draw. The two transformations also allow you to work in a variety of coordinate systems.

In This Section

Types of Coordinate Systems
Introduces coordinates systems and transformations.

Matrix Representation of Transformations
Discusses using matrices for coordinate transformations.

Global and Local Transformations
Discusses global and local transformations.

Reference

Matrix
Encapsulates a 3-by-3 affine matrix that represents a geometric transform.

Related Sections

Using Transformations in Managed GDI+
Provides a list of topics that provide more information about how to use matrix transformations.

About GDI+ Managed Code
Contains a list of topics describing the graphics constructs you can use in the .NET Framework.

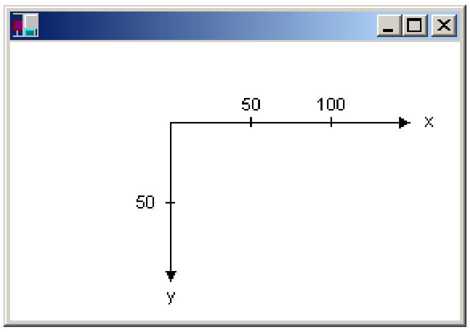
Types of Coordinate Systems

3/9/2021 • 4 minutes to read • Edit Online

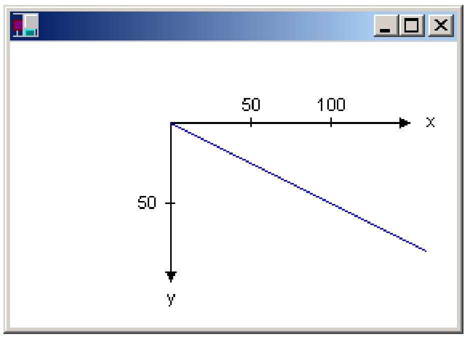
GDI+ uses three coordinate spaces: world, page, and device. World coordinates are the coordinates used to model a particular graphic world and are the coordinates you pass to methods in the .NET Framework. Page coordinates refer to the coordinate system used by a drawing surface, such as a form or control. Device coordinates are the coordinates used by the physical device being drawn on, such as a screen or sheet of paper. When you make the call `myGraphics.DrawLine(myPen, 0, 0, 160, 80)`, the points that you pass to the DrawLine method— `(0, 0)` and `(160, 80)` —are in the world coordinate space. Before GDI+ can draw the line on the screen, the coordinates pass through a sequence of transformations. One transformation, called the world transformation, converts world coordinates to page coordinates, and another transformation, called the page transformation, converts page coordinates to device coordinates.

Transforms and Coordinate Systems

Suppose you want to work with a coordinate system that has its origin in the body of the client area rather than the upper-left corner. Say, for example, that you want the origin to be 100 pixels from the left edge of the client area and 50 pixels from the top of the client area. The following illustration shows such a coordinate system.



When you make the call `myGraphics.DrawLine(myPen, 0, 0, 160, 80)`, you get the line shown in the following illustration.



The coordinates of the endpoints of your line in the three coordinate spaces are as follows:

World	(0, 0) to (160, 80)
Page	(100, 50) to (260, 130)
Device	(100, 50) to (260, 130)

Note that the page coordinate space has its origin at the upper-left corner of the client area; this will always be the case. Also note that because the unit of measure is the pixel, the device coordinates are the same as the page coordinates. If you set the unit of measure to something other than pixels (for example, inches), then the device coordinates will be different from the page coordinates.

The world transformation, which maps world coordinates to page coordinates, is held in the Transform property of the Graphics class. In the preceding example, the world transformation is a translation 100 units in the x direction and 50 units in the y direction. The following example sets the world transformation of a Graphics object and then uses that Graphics object to draw the line shown in the preceding figure:

```
myGraphics.TranslateTransform(100, 50);
myGraphics.DrawLine(myPen, 0, 0, 160, 80);
```

```
myGraphics.TranslateTransform(100, 50)
myGraphics.DrawLine(myPen, 0, 0, 160, 80)
```

The page transformation maps page coordinates to device coordinates. The Graphics class provides the PageUnit and PageScale properties for manipulating the page transformation. The Graphics class also provides two read-only properties, DpiX and DpiY, for examining the horizontal and vertical dots per inch of the display device.

You can use the PageUnit property of the Graphics class to specify a unit of measure other than the pixel.

NOTE

You cannot set the PageUnit property to World, as this is not a physical unit and will cause an exception.

The following example draws a line from (0, 0) to (2, 1), where the point (2, 1) is 2 inches to the right and 1 inch down from the point (0, 0):

```
myGraphics.PageUnit = GraphicsUnit.Inch;
myGraphics.DrawLine(myPen, 0, 0, 2, 1);
```

```
myGraphics.PageUnit = GraphicsUnit.Inch
myGraphics.DrawLine(myPen, 0, 0, 2, 1)
```

NOTE

If you don't specify a pen width when you construct your pen, the preceding example will draw a line that is one inch wide. You can specify the pen width in the second argument to the Pen constructor:

```
Pen myPen = new Pen(Color.Black, 1 / myGraphics.DpiX);
```

```
Dim myPen As New Pen(Color.Black, 1 / myGraphics.DpiX)
```

If we assume that the display device has 96 dots per inch in the horizontal direction and 96 dots per inch in the vertical direction, the endpoints of the line in the preceding example have the following coordinates in the three coordinate spaces:

World	(0, 0) to (2, 1)
Page	(0, 0) to (2, 1)
Device	(0, 0) to (192, 96)

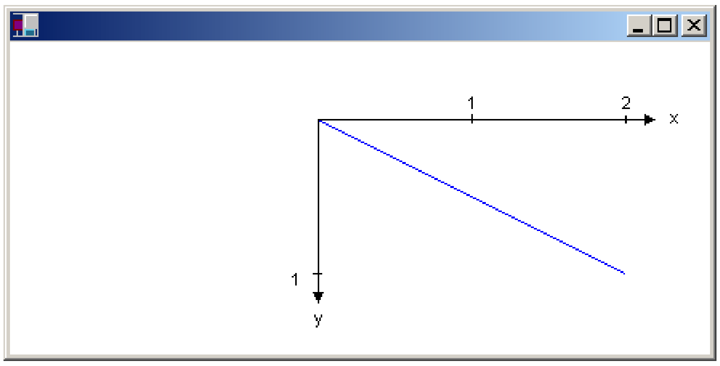
Note that because the origin of the world coordinate space is at the upper-left corner of the client area, the page coordinates are the same as the world coordinates.

You can combine the world and page transformations to achieve a variety of effects. For example, suppose you want to use inches as the unit of measure and you want the origin of your coordinate system to be 2 inches from the left edge of the client area and 1/2 inch from the top of the client area. The following example sets the world and page transformations of a Graphics object and then draws a line from (0, 0) to (2, 1):

```
myGraphics.TranslateTransform(2, 0.5f);
myGraphics.PageUnit = GraphicsUnit.Inch;
myGraphics.DrawLine(myPen, 0, 0, 2, 1);
```

```
myGraphics.TranslateTransform(2, 0.5F)
myGraphics.PageUnit = GraphicsUnit.Inch
myGraphics.DrawLine(myPen, 0, 0, 2, 1)
```

The following illustration shows the line and coordinate system.



If we assume that the display device has 96 dots per inch in the horizontal direction and 96 dots per inch in the vertical direction, the endpoints of the line in the preceding example have the following coordinates in the three coordinate spaces:

World	(0, 0) to (2, 1)
Page	(2, 0.5) to (4, 1.5)
Device	(192, 48) to (384, 144)

See also

- Coordinate Systems and Transformations
- Matrix Representation of Transformations

Matrix Representation of Transformations

11/3/2020 • 5 minutes to read • Edit Online

An m×n matrix is a set of numbers arranged in m rows and n columns. The following illustration shows several matrices.

$$\underset{2\ 3}{\begin{bmatrix} 3 & 1 & 4 \\ 2 & 5 & 0 \end{bmatrix}} \quad \underset{4\ 2}{\begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 0 & 4 \\ 5 & 6 \end{bmatrix}} \quad \underset{1\ 3}{\begin{bmatrix} 1.6 & 0.2 & 1.0 \end{bmatrix}}$$

$$\underset{2\ 2}{\begin{bmatrix} 2 & 0 \\ 0 & 3.5 \end{bmatrix}} \quad \underset{2\ 1}{\begin{bmatrix} 5 \\ 3 \end{bmatrix}} \quad \underset{3\ 3}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 40 & 20 & 1 \end{bmatrix}}$$

You can add two matrices of the same size by adding individual elements. The following illustration shows two examples of matrix addition.

$$\begin{bmatrix} 5 & 4 \end{bmatrix} + \begin{bmatrix} 20 & 30 \end{bmatrix} = \begin{bmatrix} 25 & 34 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 1 & 5 \\ 0 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 1 & 7 \\ 1 & 9 \end{bmatrix}$$

An m×n matrix can be multiplied by an n×p matrix, and the result is an m×p matrix. The number of columns in the first matrix must be the same as the number of rows in the second matrix. For example, a 4×2 matrix can be multiplied by a 2×3 matrix to produce a 4×3 matrix.

Points in the plane and rows and columns of a matrix can be thought of as vectors. For example, (2, 5) is a vector with two components, and (3, 7, 1) is a vector with three components. The dot product of two vectors is defined as follows:

(a, b) • (c, d) = ac + bd

(a, b, c) • (d, e, f) = ad + be + cf

For example, the dot product of (2, 3) and (5, 4) is (2)(5) + (3)(4) = 22. The dot product of (2, 5, 1) and (4, 3, 1) is (2)(4) + (5)(3) + (1)(1) = 24. Note that the dot product of two vectors is a number, not another vector. Also note that you can calculate the dot product only if the two vectors have the same number of components.

Let A(i, j) be the entry in matrix A in the ith row and the jth column. For example A(3, 2) is the entry in matrix A in the 3rd row and the 2nd column. Suppose A, B, and C are matrices, and AB = C. The entries of C are calculated as follows:

C(i, j) = (row i of A) • (column j of B)

The following illustration shows several examples of matrix multiplication.

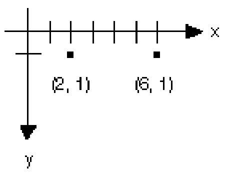
$$\underset{1\times 2}{\begin{bmatrix} 2 & 3 \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} 2 \\ 4 \end{bmatrix}} = \underset{1\times 1}{\begin{bmatrix} 16 \end{bmatrix}}$$

$$\underset{1\times 3}{\begin{bmatrix} 1 & 3 & 2 \end{bmatrix}} \underset{3\times 2}{\begin{bmatrix} 1 & 0 \\ 2 & 4 \\ 5 & 1 \end{bmatrix}} = \underset{1\times 2}{\begin{bmatrix} 17 & 14 \end{bmatrix}}$$

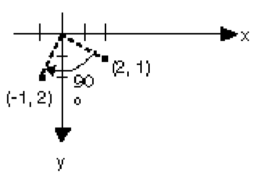
$$\underset{2\times 3}{\begin{bmatrix} 2 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix}} \underset{3\times 3}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 3 & 1 \end{bmatrix}} = \underset{2\times 3}{\begin{bmatrix} 4 & 13 & 1 \\ 6 & 9 & 1 \end{bmatrix}}$$

If you think of a point in a plane as a 1×2 matrix, you can transform that point by multiplying it by a 2×2 matrix. The following illustration shows several transformations applied to the point (2, 1).

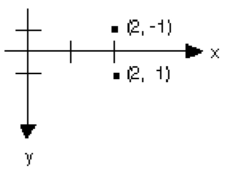
Scale $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 1 \end{bmatrix}$



Rotate 90 ° $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 2 \end{bmatrix}$

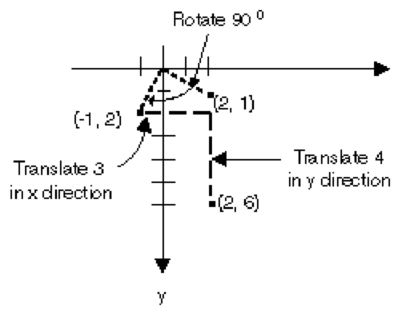


Reflect across x-axis $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix}$



All of the transformations shown in the preceding figure are linear transformations. Certain other transformations, such as translation, are not linear, and cannot be expressed as multiplication by a 2×2 matrix. Suppose you want to start with the point (2, 1), rotate it 90 degrees, translate it 3 units in the x direction, and translate it 4 units in the y direction. You can accomplish this by using a matrix multiplication followed by a matrix addition.

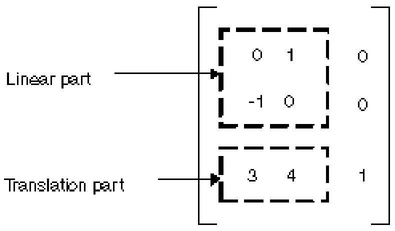
$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 6 \end{bmatrix}$$



A linear transformation (multiplication by a 2×2 matrix) followed by a translation (addition of a 1×2 matrix) is called an affine transformation. An alternative to storing an affine transformation in a pair of matrices (one for the linear part and one for the translation) is to store the entire transformation in a 3×3 matrix. To make this work, a point in the plane must be stored in a 1×3 matrix with a dummy 3rd coordinate. The usual technique is to make all 3rd coordinates equal to 1. For example, the point (2, 1) is represented by the matrix [2 1 1]. The following illustration shows an affine transformation (rotate 90 degrees; translate 3 units in the x direction, 4 units in the y direction) expressed as multiplication by a single 3×3 matrix.

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 1 \end{bmatrix}$$

In the preceding example, the point (2, 1) is mapped to the point (2, 6). Note that the third column of the 3×3 matrix contains the numbers 0, 0, 1. This will always be the case for the 3×3 matrix of an affine transformation. The important numbers are the six numbers in columns 1 and 2. The upper-left 2×2 portion of the matrix represents the linear part of the transformation, and the first two entries in the 3rd row represent the translation.



In GDI+ you can store an affine transformation in a Matrix object. Because the third column of a matrix that represents an affine transformation is always (0, 0, 1), you specify only the six numbers in the first two columns when you construct a Matrix object. The statement `Matrix myMatrix = new Matrix(0, 1, -1, 0, 3, 4)` constructs the matrix shown in the preceding figure.

Composite Transformations

A composite transformation is a sequence of transformations, one followed by the other. Consider the matrices and transformations in the following list:

Matrix A	Rotate 90 degrees
Matrix B	Scale by a factor of 2 in the x direction
Matrix C	Translate 3 units in the y direction

If we start with the point (2, 1) — represented by the matrix [2 1 1] — and multiply by A, then B, then C, the point (2, 1) will undergo the three transformations in the order listed.

[2 1 1]ABC = [-2 5 1]

Rather than store the three parts of the composite transformation in three separate matrices, you can multiply A, B, and C together to get a single 3×3 matrix that stores the entire composite transformation. Suppose ABC = D. Then a point multiplied by D gives the same result as a point multiplied by A, then B, then C.

[2 1 1]D = [-2 5 1]

The following illustration shows the matrices A, B, C, and D.

$$\underset{A}{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{B}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{C}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix}} \underset{=}{=} \underset{D}{\begin{bmatrix} 0 & 1 & 0 \\ -2 & 0 & 0 \\ 0 & 3 & 1 \end{bmatrix}}$$

The fact that the matrix of a composite transformation can be formed by multiplying the individual transformation matrices means that any sequence of affine transformations can be stored in a single Matrix object.

Caution

The order of a composite transformation is important. In general, rotate, then scale, then translate is not the same as scale, then rotate, then translate. Similarly, the order of matrix multiplication is important. In general, ABC is not the same as BAC.

The Matrix class provides several methods for building a composite transformation: Multiply, Rotate, RotateAt, Scale, Shear, and Translate. The following example creates the matrix of a composite transformation that first rotates 30 degrees, then scales by a factor of 2 in the y direction, and then translates 5 units in the x direction:

```
Matrix myMatrix = new Matrix();
myMatrix.Rotate(30);
myMatrix.Scale(1, 2, MatrixOrder.Append);
myMatrix.Translate(5, 0, MatrixOrder.Append);
```

```
Dim myMatrix As New Matrix()
myMatrix.Rotate(30)
myMatrix.Scale(1, 2, MatrixOrder.Append)
myMatrix.Translate(5, 0, MatrixOrder.Append)
```

The following illustration shows the matrix.

$$\begin{bmatrix} \cos 30^\circ & 2\sin 30^\circ & 0 \\ -\sin 30^\circ & 2\cos 30^\circ & 0 \\ 5 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 0.866 & 1.0 & 0 \\ -0.5 & 1.73 & 0 \\ 5 & 0 & 1 \end{bmatrix}$$

See also

- Coordinate Systems and Transformations
- Using Transformations in Managed GDI+

Global and Local Transformations

11/3/2020 • 3 minutes to read • Edit Online

A global transformation is a transformation that applies to every item drawn by a given Graphics object. In contrast, a local transformation is a transformation that applies to a specific item to be drawn.

Global Transformations

To create a global transformation, construct a Graphics object, and then manipulate its Transform property. The Transform property is a Matrix object, so it can hold any sequence of affine transformations. The transformation stored in the Transform property is called the world transformation. The Graphics class provides several methods for building up a composite world transformation: MultiplyTransform, RotateTransform, ScaleTransform, and TranslateTransform. The following example draws an ellipse twice: once before creating a world transformation and once after. The transformation first scales by a factor of 0.5 in the y direction, then translates 50 units in the x direction, and then rotates 30 degrees.

```
myGraphics.DrawEllipse(myPen, 0, 0, 100, 50);
myGraphics.ScaleTransform(1, 0.5f);
myGraphics.TranslateTransform(50, 0, MatrixOrder.Append);
myGraphics.RotateTransform(30, MatrixOrder.Append);
myGraphics.DrawEllipse(myPen, 0, 0, 100, 50);
```

```
myGraphics.DrawEllipse(myPen, 0, 0, 100, 50)
myGraphics.ScaleTransform(1, 0.5F)
myGraphics.TranslateTransform(50, 0, MatrixOrder.Append)
myGraphics.RotateTransform(30, MatrixOrder.Append)
myGraphics.DrawEllipse(myPen, 0, 0, 100, 50)
```

The following illustration shows the matrices involved in the transformation.

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Scale}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 50 & 0 & 1 \end{bmatrix}}_{\text{Translate}} \underbrace{\begin{bmatrix} \cos 30° & \sin 30° & 0 \\ -\sin 30° & \cos 30° & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Rotate}} = \begin{bmatrix} \cos 30° & \sin 30° & 0 \\ -0.5\sin 30° & 0.5\cos 30° & 0 \\ 50\cos 30° & 50\sin 30° & 1 \end{bmatrix}$$

NOTE

In the preceding example, the ellipse is rotated about the origin of the coordinate system, which is at the upper-left corner of the client area. This produces a different result than rotating the ellipse about its own center.

Local Transformations

A local transformation applies to a specific item to be drawn. For example, a GraphicsPath object has a Transform method that allows you to transform the data points of that path. The following example draws a rectangle with no transformation and a path with a rotation transformation. (Assume that there is no world transformation.)

```
Matrix myMatrix = new Matrix();
myMatrix.Rotate(45);
myGraphicsPath.Transform(myMatrix);
myGraphics.DrawRectangle(myPen, 10, 10, 100, 50);
myGraphics.DrawPath(myPen, myGraphicsPath);
```

```
Dim myMatrix As New Matrix()
myMatrix.Rotate(45)
myGraphicsPath.Transform(myMatrix)
myGraphics.DrawRectangle(myPen, 10, 10, 100, 50)
myGraphics.DrawPath(myPen, myGraphicsPath)
```

You can combine the world transformation with local transformations to achieve a variety of results. For example, you can use the world transformation to revise the coordinate system and use local transformations to rotate and scale objects drawn on the new coordinate system.

Suppose you want a coordinate system that has its origin 200 pixels from the left edge of the client area and 150 pixels from the top of the client area. Furthermore, assume that you want the unit of measure to be the pixel, with the x-axis pointing to the right and the y-axis pointing up. The default coordinate system has the y-axis pointing down, so you need to perform a reflection across the horizontal axis. The following illustration shows the matrix of such a reflection.

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

Next, assume you need to perform a translation 200 units to the right and 150 units down.

The following example establishes the coordinate system just described by setting the world transformation of a Graphics object.

```
Matrix myMatrix = new Matrix(1, 0, 0, -1, 0, 0);
myGraphics.Transform = myMatrix;
myGraphics.TranslateTransform(200, 150, MatrixOrder.Append);
```

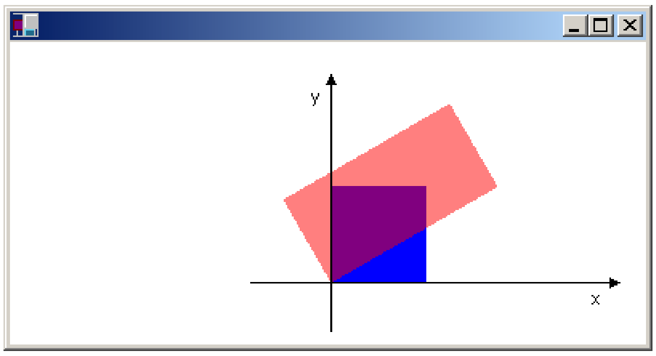
```
Dim myMatrix As New Matrix(1, 0, 0, -1, 0, 0)
myGraphics.Transform = myMatrix
myGraphics.TranslateTransform(200, 150, MatrixOrder.Append)
```

The following code (placed at the end of the preceding example) creates a path that consists of a single rectangle with its lower-left corner at the origin of the new coordinate system. The rectangle is filled once with no local transformation and once with a local transformation. The local transformation consists of a horizontal scaling by a factor of 2 followed by a 30-degree rotation.

```
// Create the path.
GraphicsPath myGraphicsPath = new GraphicsPath();
Rectangle myRectangle = new Rectangle(0, 0, 60, 60);
myGraphicsPath.AddRectangle(myRectangle);

// Fill the path on the new coordinate system.
// No local transformation
myGraphics.FillPath(mySolidBrush1, myGraphicsPath);

// Set the local transformation of the GraphicsPath object.
Matrix myPathMatrix = new Matrix();
myPathMatrix.Scale(2, 1);
myPathMatrix.Rotate(30, MatrixOrder.Append);
myGraphicsPath.Transform(myPathMatrix);

// Fill the transformed path on the new coordinate system.
myGraphics.FillPath(mySolidBrush2, myGraphicsPath);
```

```
' Create the path.
Dim myGraphicsPath As New GraphicsPath()
Dim myRectangle As New Rectangle(0, 0, 60, 60)
myGraphicsPath.AddRectangle(myRectangle)

' Fill the path on the new coordinate system.
' No local transformation
myGraphics.FillPath(mySolidBrush1, myGraphicsPath)

' Set the local transformation of the GraphicsPath object.
Dim myPathMatrix As New Matrix()
myPathMatrix.Scale(2, 1)
myPathMatrix.Rotate(30, MatrixOrder.Append)
myGraphicsPath.Transform(myPathMatrix)

' Fill the transformed path on the new coordinate system.
myGraphics.FillPath(mySolidBrush2, myGraphicsPath)
```

The following illustration shows the new coordinate system and the two rectangles.



See also

- Coordinate Systems and Transformations
- Using Transformations in Managed GDI+

Using Managed Graphics Classes

11/3/2020 • 2 minutes to read • Edit Online

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.

Getting Started with Graphics Programming

11/3/2020 • 2 minutes to read • Edit Online

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

11/3/2020 • 4 minutes to read • Edit Online

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:

```
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
```

```
private void Form1_Paint(object sender,
   System.Windows.Forms.PaintEventArgs pe)
{
   // Declares the Graphics object and sets it to the Graphics object
   // supplied in the PaintEventArgs.
   Graphics g = pe.Graphics;
   // Insert code to paint the form here.
}
```

```
private:
   void Form1_Paint(System::Object ^ sender,
      System::Windows::Forms::PaintEventArgs ^ pe)
   {
      // Declares the Graphics object and sets it to the Graphics object
      // supplied in the PaintEventArgs.
      Graphics ^ g = pe->Graphics;
      // Insert code to paint the form here.
   }
```

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.

```
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
```

```
Graphics g;
// Sets g to a graphics object representing the drawing surface of the
// control or form g is a member of.
g = this.CreateGraphics();
```

```
Graphics ^ g;
// Sets g to a graphics object representing the drawing surface of the
// control or form g is a member of.
g = this->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:

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

  ```
  Bitmap myBitmap = new Bitmap(@"C:\Documents and
     Settings\Joe\Pics\myPic.bmp");
  Graphics g = Graphics.FromImage(myBitmap);
  ```

  ```
  Bitmap ^ myBitmap = gcnew
     Bitmap("D:\\Documents and Settings\\Joe\\Pics\\myPic.bmp");
  Graphics ^ g = 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

11/3/2020 • 2 minutes to read • Edit Online

This example creates a Pen object.

Example

```
System::Drawing::Pen^ myPen;
myPen = gcnew System::Drawing::Pen(System::Drawing::Color::Tomato);
```

```
System.Drawing.Pen myPen;
myPen = new System.Drawing.Pen(System.Drawing.Color.Tomato);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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

Example

```
myPen->Color = System::Drawing::Color::PeachPuff;
```

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

```
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

11/3/2020 • 2 minutes to read • Edit Online

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

Example

```
System::Drawing::SolidBrush^ myBrush =
    gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);
System::Drawing::Graphics^ formGraphics;
formGraphics = this->CreateGraphics();
formGraphics->FillEllipse(myBrush, Rectangle(0, 0, 200, 300));
delete myBrush;
delete formGraphics;
```

```
System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);
System.Drawing.Graphics formGraphics;
formGraphics = this.CreateGraphics();
formGraphics.FillEllipse(myBrush, new Rectangle(0, 0, 200, 300));
myBrush.Dispose();
formGraphics.Dispose();
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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

```
Pen pen = new Pen(Color.FromArgb(255, 0, 0, 0));
e.Graphics.DrawLine(pen, 20, 10, 300, 100);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

This example draws outlined ellipses and rectangles on a form.

Example

```
private:
    void DrawEllipse()
    {
        System::Drawing::Pen^ myPen =
            gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Graphics^ formGraphics;
        formGraphics = this->CreateGraphics();
        formGraphics->DrawEllipse(myPen, Rectangle(0, 0, 200, 300));
        delete myPen;
        delete formGraphics;
    }

private:
    void DrawRectangle()
    {
        System::Drawing::Pen^ myPen =
            gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Graphics^ formGraphics;
        formGraphics = this->CreateGraphics();
        formGraphics->DrawRectangle(myPen, Rectangle(0, 0, 200, 300));
        delete myPen;
        delete formGraphics;
    }
```

```
private void DrawEllipse()
{
    System.Drawing.Pen myPen = new System.Drawing.Pen(System.Drawing.Color.Red);
    System.Drawing.Graphics formGraphics;
    formGraphics = this.CreateGraphics();
    formGraphics.DrawEllipse(myPen, new Rectangle(0, 0, 200, 300));
    myPen.Dispose();
    formGraphics.Dispose();
}

private void DrawRectangle()
{
    System.Drawing.Pen myPen = new System.Drawing.Pen(System.Drawing.Color.Red);
    System.Drawing.Graphics formGraphics;
    formGraphics = this.CreateGraphics();
    formGraphics.DrawRectangle(myPen, new Rectangle(0, 0, 200, 300));
    myPen.Dispose();
    formGraphics.Dispose();
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

This example draws a filled rectangle on a form.

Example

```
System::Drawing::SolidBrush^ myBrush =
    gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);
System::Drawing::Graphics^ formGraphics;
formGraphics = this->CreateGraphics();
formGraphics->FillRectangle(myBrush, Rectangle(0, 0, 200, 300));
delete myBrush;
delete formGraphics;
```

```
System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);
System.Drawing.Graphics formGraphics;
formGraphics = this.CreateGraphics();
formGraphics.FillRectangle(myBrush, new Rectangle(0, 0, 200, 300));
myBrush.Dispose();
formGraphics.Dispose();
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

This example draws a filled ellipse on a form.

Example

```
System::Drawing::SolidBrush^ myBrush =
    gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);
System::Drawing::Graphics^ formGraphics;
formGraphics = this->CreateGraphics();
formGraphics->FillEllipse(myBrush, Rectangle(0, 0, 200, 300));
delete myBrush;
delete formGraphics;
```

```
System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);
System.Drawing.Graphics formGraphics;
formGraphics = this.CreateGraphics();
formGraphics.FillEllipse(myBrush, new Rectangle(0, 0, 200, 300));
myBrush.Dispose();
formGraphics.Dispose();
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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

```
public:
    void DrawString()
    {
        System::Drawing::Graphics^ formGraphics = this->CreateGraphics();
        String^ drawString = "Sample Text";
        System::Drawing::Font^ drawFont =
            gcnew System::Drawing::Font("Arial", 16);
        System::Drawing::SolidBrush^ drawBrush = gcnew
            System::Drawing::SolidBrush(System::Drawing::Color::Black);
        float x = 150.0F;
        float y = 50.0F;
        System::Drawing::StringFormat^ drawFormat =
            gcnew System::Drawing::StringFormat();
        formGraphics->DrawString(drawString, drawFont, drawBrush, x,
            y, drawFormat);
        delete drawFont;
        delete drawBrush;
        delete formGraphics;
    }
```

```
public void DrawString()
{
    System.Drawing.Graphics formGraphics = this.CreateGraphics();
    string drawString = "Sample Text";
    System.Drawing.Font drawFont = new System.Drawing.Font("Arial", 16);
    System.Drawing.SolidBrush drawBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Black);
    float x = 150.0F;
    float y = 50.0F;
    System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
    formGraphics.DrawString(drawString, drawFont, drawBrush, x, y, drawFormat);
    drawFont.Dispose();
    drawBrush.Dispose();
    formGraphics.Dispose();
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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

Example

```
public:
    void DrawVerticalString()
    {
        System::Drawing::Graphics^ formGraphics = this->CreateGraphics();
        String^ drawString = "Sample Text";
        System::Drawing::Font^ drawFont =
            gcnew System::Drawing::Font("Arial", 16);
        System::Drawing::SolidBrush^ drawBrush = gcnew
            System::Drawing::SolidBrush(System::Drawing::Color::Black);
        float x = 150.0F;
        float y = 50.0F;
        System::Drawing::StringFormat^ drawFormat =
            gcnew System::Drawing::StringFormat();
        drawFormat->FormatFlags = StringFormatFlags::DirectionVertical;
        formGraphics->DrawString(drawString, drawFont, drawBrush, x,
            y, drawFormat);
        delete drawFont;
        delete drawBrush;
        delete formGraphics;
    }
```

```
public void DrawVerticalString()
{
    System.Drawing.Graphics formGraphics = this.CreateGraphics();
    string drawString = "Sample Text";
    System.Drawing.Font drawFont = new System.Drawing.Font("Arial", 16);
    System.Drawing.SolidBrush drawBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Black);
    float x = 150.0F;
    float y = 50.0F;
    System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
    drawFormat.FormatFlags = StringFormatFlags.DirectionVertical;
    formGraphics.DrawString(drawString, drawFont, drawBrush, x, y, drawFormat);
    drawFont.Dispose();
    drawBrush.Dispose();
    formGraphics.Dispose();
}
```

```
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+

11/3/2020 • 2 minutes to read • Edit Online

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:

```
' 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))
```

```
// Uses the System.Environment.GetFolderPath to get the path to the
// current user's MyPictures folder.
Bitmap myBitmap = new Bitmap
   (System.Environment.GetFolderPath
      (System.Environment.SpecialFolder.MyPictures));
```

```
// Uses the System.Environment.GetFolderPath to get the path to the
// current user's MyPictures folder.
Bitmap^ myBitmap = gcnew 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.

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

```
// Creates a Graphics object that represents the drawing surface of
// Button1.
Graphics g = Button1.CreateGraphics();
```

```
// Creates a Graphics object that represents the drawing surface of
// Button1.
Graphics^ g = 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.

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

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

```
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

How to: Create a Shaped Windows Form

11/3/2020 • 2 minutes to read • Edit Online

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

Example

```
protected:
    virtual void OnPaint(
        System::Windows::Forms::PaintEventArgs^ e) override
    {
        System::Drawing::Drawing2D::GraphicsPath^ shape =
            gcnew System::Drawing::Drawing2D::GraphicsPath();
        shape->AddEllipse(0, 0, this->Width, this->Height);
        this->Region = gcnew System::Drawing::Region(shape);
    }
```

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
    System.Drawing.Drawing2D.GraphicsPath shape = new System.Drawing.Drawing2D.GraphicsPath();
    shape.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new System.Drawing.Region(shape);
}
```

```
 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

11/3/2020 • 2 minutes to read • Edit Online

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 discernible performance gains when using the code below, it is something to keep in mind when working with more complex graphics-manipulation code.

Example

```
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
```

```
private void Form1_Paint(System.Object sender,
    System.Windows.Forms.PaintEventArgs e)
        {
        e.Graphics.FillEllipse(Brushes.DarkBlue, new
            Rectangle(10,10,60,60));
        e.Graphics.FillRectangle(Brushes.Khaki, new
            Rectangle(20,30,60,10));
        e.Graphics.CopyFromScreen(new Point(10, 10), new Point(100, 100),
            new Size(70, 70));
}
```

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

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Use a Pen to Draw Lines

11/3/2020 • 2 minutes to read • Edit Online

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.

```
Pen pen = new Pen(Color.FromArgb(255, 0, 0, 0));
e.Graphics.DrawLine(pen, 20, 10, 300, 100);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

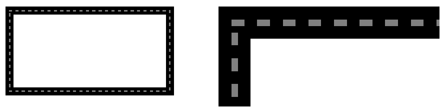
To draw rectangles, you need a Graphics object and a Pen object. The Graphics object provides the DrawRectangle method, and the Pen object stores features of the line, such as color and width.

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.



```
Pen blackPen = new Pen(Color.FromArgb(255, 0, 0, 0), 5);
e.Graphics.DrawRectangle(blackPen, 10, 10, 100, 50);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

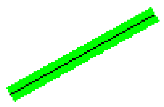
A theoretical line has a width of 0. When you draw a line that is 1 pixel wide, the pixels are centered on the theoretical line. If you draw a line that is more than one pixel wide, the pixels are either centered on the theoretical line or appear to one side of the theoretical line. You can set the pen 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.

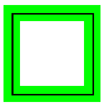
```
Pen blackPen = new Pen(Color.FromArgb(255, 0, 0, 0), 1);
Pen greenPen = new Pen(Color.FromArgb(255, 0, 255, 0), 10);
greenPen.Alignment = PenAlignment.Center;

// 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);
```

```
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:



```
Pen blackPen = new Pen(Color.FromArgb(255, 0, 0, 0), 1);
Pen greenPen = 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);
```

```
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)
```

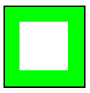
To create an inset pen

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

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

```
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

11/3/2020 • 2 minutes to read • Edit Online

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:

```
Pen pen = 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);
```

```
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

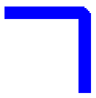
- System.Drawing.Pen
- System.Drawing.Drawing2D.LineCap
- Graphics and Drawing in Windows Forms
- Using a Pen to Draw Lines and Shapes

How to: Join Lines

11/3/2020 • 2 minutes to read • Edit Online

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.

```
GraphicsPath path = new GraphicsPath();
Pen penJoin = 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);
```

```
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

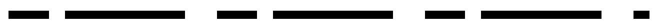
11/3/2020 • 2 minutes to read • Edit Online

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).



```
float[] dashValues = { 5, 2, 15, 4 };
Pen blackPen = new Pen(Color.Black, 5);
blackPen.DashPattern = dashValues;
e.Graphics.DrawLine(blackPen, new Point(5, 5), new Point(405, 5));
```

```
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

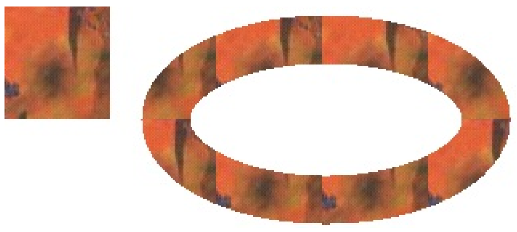
11/3/2020 • 2 minutes to read • Edit Online

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:



```
Bitmap bitmap = new Bitmap("Texture1.jpg");
TextureBrush tBrush = new TextureBrush(bitmap);
Pen texturedPen = new Pen(tBrush, 30);

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

```
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

Using a Brush to Fill Shapes

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Fill a Shape with a Solid Color

11/3/2020 • 2 minutes to read • Edit Online

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.

```
SolidBrush solidBrush = new SolidBrush(
   Color.FromArgb(255, 255, 0, 0));
e.Graphics.FillEllipse(solidBrush, 0, 0, 100, 60);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

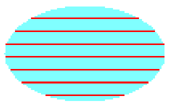
A hatch pattern is made from two colors: one for the background and one for the lines that form the pattern over the background. To fill a closed shape with a hatch pattern, use a HatchBrush object. The following example demonstrates how to fill an ellipse with a hatch pattern:

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.



```
HatchBrush hBrush = new HatchBrush(
   HatchStyle.Horizontal,
   Color.Red,
   Color.FromArgb(255, 128, 255, 255));
e.Graphics.FillEllipse(hBrush, 0, 0, 100, 60);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
Image image = new Bitmap("ImageFile.jpg");
TextureBrush tBrush = 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));
```

```
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

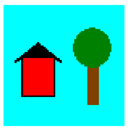
11/3/2020 • 3 minutes to read • Edit Online

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.

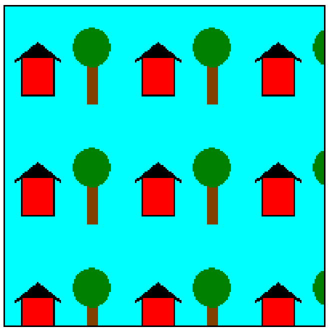
The wrap mode property of the TextureBrush object determines how the image is oriented as it is repeated in a rectangular grid. You can make all the tiles in the grid have the same orientation, or you can make the image flip from one grid position to the next. The flipping can be horizontal, vertical,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

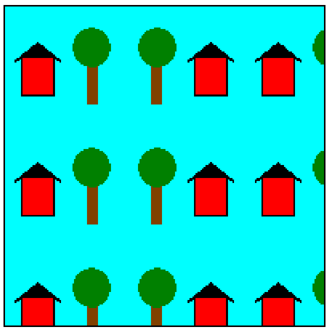


```
Image image = new Bitmap("HouseAndTree.gif");
TextureBrush tBrush = new TextureBrush(image);
Pen blackPen = new Pen(Color.Black);
e.Graphics.FillRectangle(tBrush, new Rectangle(0, 0, 200, 200));
e.Graphics.DrawRectangle(blackPen, new Rectangle(0, 0, 200, 200));
```

```
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.



```
Image image = new Bitmap("HouseAndTree.gif");
TextureBrush tBrush = new TextureBrush(image);
Pen blackPen = 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));
```

```
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

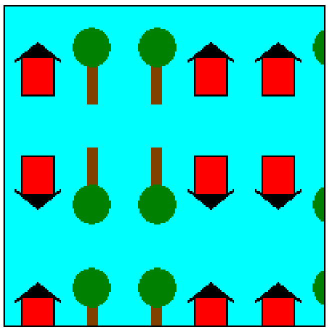
- This example uses the same 75×75 image to fill a 200×200 rectangle. The wrap mode is set to flip the image vertically.

  ```
  Image image = new Bitmap("HouseAndTree.gif");
  TextureBrush tBrush = new TextureBrush(image);
  Pen blackPen = 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));
  ```

  ```
  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.



```
Image image = new Bitmap("HouseAndTree.gif");
TextureBrush tBrush = new TextureBrush(image);
Pen blackPen = 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));
```

```
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

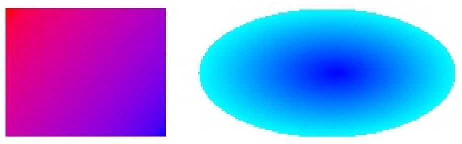
Using a Gradient Brush to Fill Shapes

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Create a Linear Gradient

11/3/2020 • 4 minutes to read • Edit Online

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.

NOTE

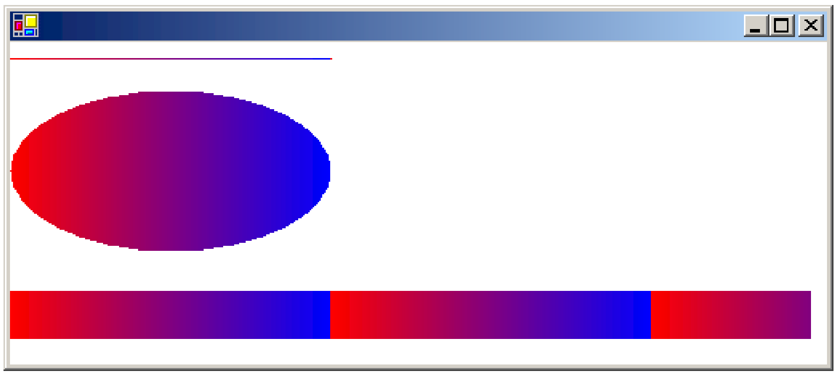
The examples in this article are methods that are called from a control's Paint event handler.

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 (0, 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.

```
public void UseHorizontalLinearGradients(PaintEventArgs e)
{
    LinearGradientBrush linGrBrush = new LinearGradientBrush(
       new Point(0, 10),
       new Point(200, 10),
       Color.FromArgb(255, 255, 0, 0),   // Opaque red
       Color.FromArgb(255, 0, 0, 255));  // Opaque blue

    Pen pen = 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);
}
```

```
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.

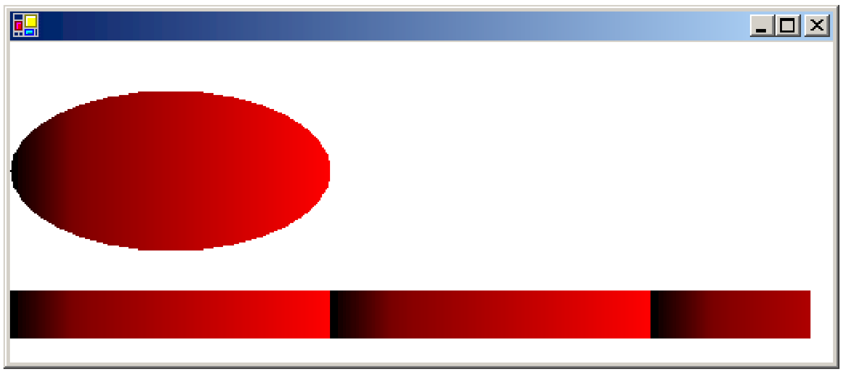
GDI+ allows you to adjust the way a color varies from one edge of a gradient to the other. Suppose you want to create a gradient brush that changes from black to red according to the following table.

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 LinearGradientBrush.Blend property 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.

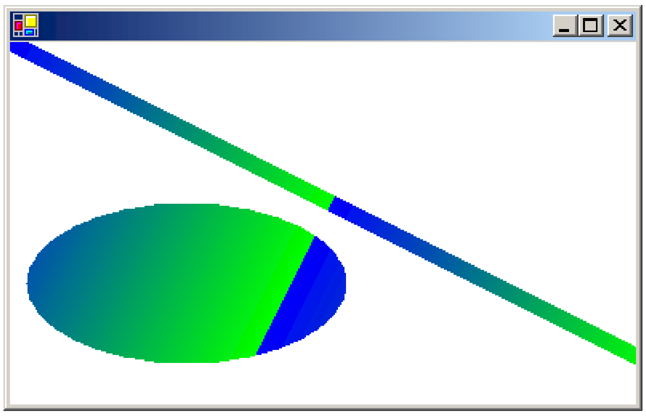
```
public void CustomizeLinearGradients(PaintEventArgs e)
{
    LinearGradientBrush linGrBrush = new LinearGradientBrush(
       new Point(0, 10),
       new Point(200, 10),
       Color.FromArgb(255, 0, 0, 0),     // Opaque black
       Color.FromArgb(255, 255, 0, 0));  // Opaque red

    float[] relativeIntensities = { 0.0f, 0.5f, 1.0f };
    float[] relativePositions = { 0.0f, 0.2f, 1.0f };

    //Create a Blend object and assign it to linGrBrush.
    Blend blend = 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);
}
```

```
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.

```
public void CreateDiagonalLinearGradients(PaintEventArgs e)
{
    LinearGradientBrush linGrBrush = new LinearGradientBrush(
       new Point(0, 0),
       new Point(200, 100),
       Color.FromArgb(255, 0, 0, 255),   // opaque blue
       Color.FromArgb(255, 0, 255, 0));  // opaque green

    Pen pen = new Pen(linGrBrush, 10);

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

```
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

11/3/2020 • 13 minutes to read • Edit Online

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.

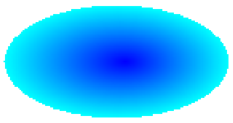
NOTE

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.

The examples in this article are methods that are called from a control's Paint event handler.

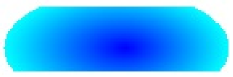
To fill an ellipse with a path gradient

- 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 Graphics.FillEllipse call in the following code to `e.Graphics.FillRectangle(pthGrBrush, 0, 10, 200, 40)`:



```
public void FillEllipseWithPathGradient(PaintEventArgs e)
{
    // Create a path that consists of a single ellipse.
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, 140, 70);

    // Use the path to construct a brush.
    PathGradientBrush pthGrBrush = 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.
    Color[] colors = { Color.FromArgb(255, 0, 255, 255) };
    pthGrBrush.SurroundColors = colors;

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

```
' 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.

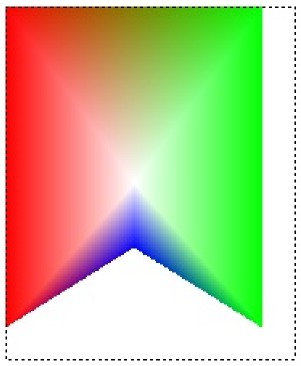
To specify points on the boundary

- 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.

```
public void ConstructBrushFromStarShapedPath(PaintEventArgs e)
{
    // Put the points of a polygon in an array.
    Point[] points = {
       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.
    GraphicsPath path = new GraphicsPath();
    path.AddLines(points);

    // Use the path to construct a path gradient brush.
    PathGradientBrush pthGrBrush = 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.
    Color[] colors = {
       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);
}
```

```
' 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:



```
public void DrawPathGradentWthoutGraphicsPath(PaintEventArgs e)
{
    // Construct a path gradient brush based on an array of points.
    PointF[] ptsF = {
       new PointF(0, 0),
       new PointF(160, 0),
       new PointF(160, 200),
       new PointF(80, 150),
       new PointF(0, 200)};

    PathGradientBrush pBrush = 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.
    Color[] colors = {
       Color.FromArgb(255, 255, 0, 0),  // (0, 0) red
       Color.FromArgb(255, 0, 255, 0),  // (160, 0) green
       Color.FromArgb(255, 0, 255, 0),  // (160, 200) green
       Color.FromArgb(255, 0, 0, 255),  // (80, 150) blue
       Color.FromArgb(255, 255, 0, 0)}; // (0, 200) red

    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));
}
```

```
' 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

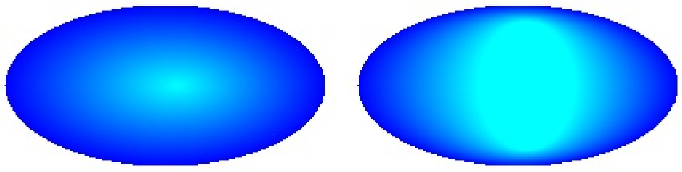
- One way to customize a path gradient brush is to set its FocusScales property. The focus scales specify an inner path that lies inside the main path. The center color is displayed everywhere inside that inner path rather than only at the center point.

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.



```
public void CustomizePathGradientBrush(PaintEventArgs e)
{
    // Create a path that consists of a single ellipse.
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, 200, 100);

    // Create a path gradient brush based on the elliptical path.
    PathGradientBrush pthGrBrush = new PathGradientBrush(path);

    // Set the color along the entire boundary to blue.
    Color[] color = { Color.Blue };
    pthGrBrush.SurroundColors = color;

    // 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);
}
```

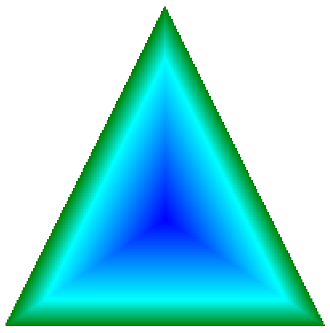
```
' Create a path that consists of a single ellipse.
Dim path As New GraphicsPath()
path.AddEllipse(0, 0, 200, 100)

' 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.



```
public void CustomizeWithInterpolation(PaintEventArgs e)
{
    // Vertices of the outer triangle
    Point[] points = {
       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.
    PathGradientBrush pthGrBrush = new PathGradientBrush(points);

    Color[] colors = {
       Color.FromArgb(255, 0, 128, 0),    // dark green
       Color.FromArgb(255, 0, 255, 255),  // aqua
       Color.FromArgb(255, 0, 0, 255)};   // blue

    float[] relativePositions = {
       0f,       // Dark green is at the boundary of the triangle.
       0.4f,     // Aqua is 40 percent of the way from the boundary
                 // to the center point.
       1.0f};    // Blue is at the center point.

    ColorBlend colorBlend = 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);
}
```

```
' 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

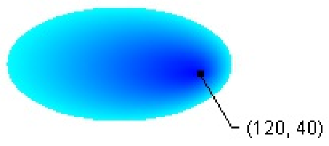
- By default, the center point of a path gradient brush is at the centroid of the path used to construct the brush. You can change the location of the center point by setting the CenterPoint property of the PathGradientBrush class.

  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).

```
public void SetCenterPoint(PaintEventArgs e)
{
    // Create a path that consists of a single ellipse.
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, 140, 70);

    // Use the path to construct a brush.
    PathGradientBrush pthGrBrush = 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.
    Color[] colors = { Color.FromArgb(255, 0, 255, 255) };
    pthGrBrush.SurroundColors = colors;

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

```
' 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:

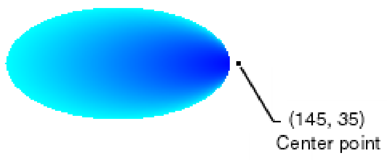


- You can set the center point of a path gradient brush to a location outside the path that was used to construct the brush. The following example replaces the call to set the CenterPoint property in the preceding code.

```
public void SetCenterPointOutsidePath(PaintEventArgs e)
{
    // Create a path that consists of a single ellipse.
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, 140, 70);

    // Use the path to construct a brush.
    PathGradientBrush pthGrBrush = new PathGradientBrush(path);

    // Set the center point to a location that is not
    // the centroid of the path.
    pthGrBrush.CenterPoint = new PointF(145, 35);

    // 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.
    Color[] colors = { Color.FromArgb(255, 0, 255, 255) };
    pthGrBrush.SurroundColors = colors;

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

```
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

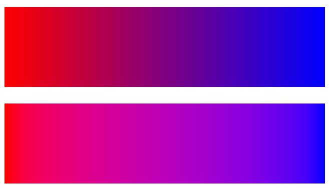
11/3/2020 • 2 minutes to read • Edit Online

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 following example is a method that is called from a control's Paint event handler. 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.



```
public void FillTwoRectangles(PaintEventArgs e)
{
   LinearGradientBrush linGrBrush = 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);
}
```

```
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

- System.Drawing.Drawing2D.LinearGradientBrush
- Using a Gradient Brush to Fill Shapes

Working with Images, Bitmaps, Icons, and Metafiles

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Draw an Existing Bitmap to the Screen

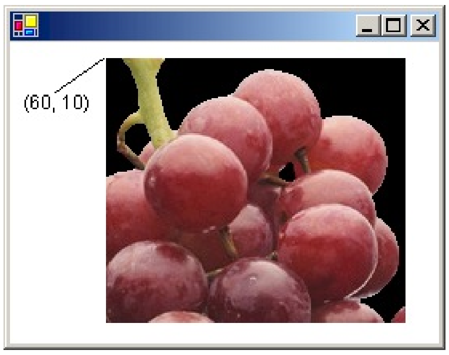
11/3/2020 • 2 minutes to read • Edit Online

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:



```
Bitmap bitmap = new Bitmap("Grapes.jpg");
e.Graphics.DrawImage(bitmap, 60, 10);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

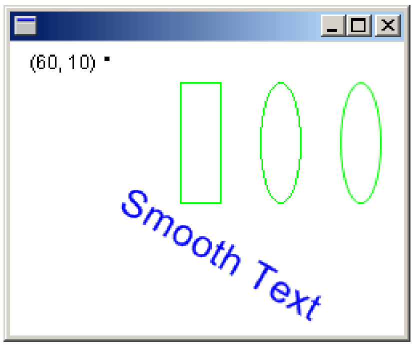
The Metafile class, which inherits from the Image class, provides methods for recording, displaying, and examining vector images.

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.



```
Metafile metafile = new Metafile("SampleMetafile.emf");
e.Graphics.DrawImage(metafile, 60, 10);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

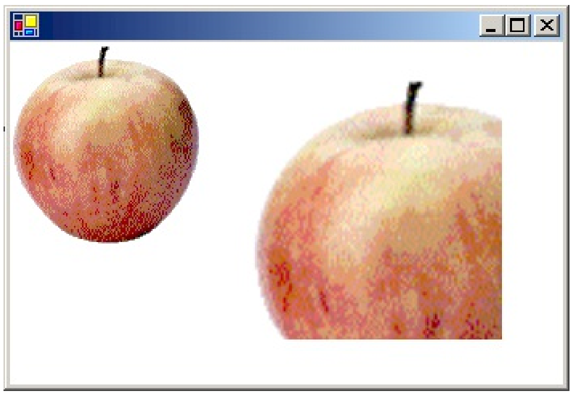
Example

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.



```
Image image = 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).
int width = image.Width;
int height = image.Height;
RectangleF destinationRect = 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.
RectangleF sourceRect = new RectangleF(0, 0, .75f * width, .75f * height);
e.Graphics.DrawImage(
    image,
    destinationRect,
    sourceRect,
    GraphicsUnit.Pixel);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

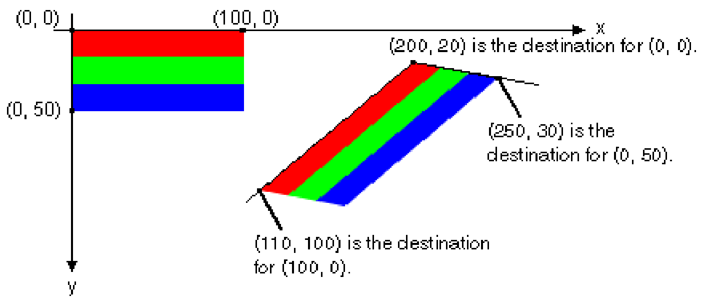
You can rotate, reflect, and skew an image by specifying destination points for the upper-left, upper-right, and lower-left corners of the original image. The three destination points determine an affine transformation that maps the original rectangular image to a parallelogram.

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.

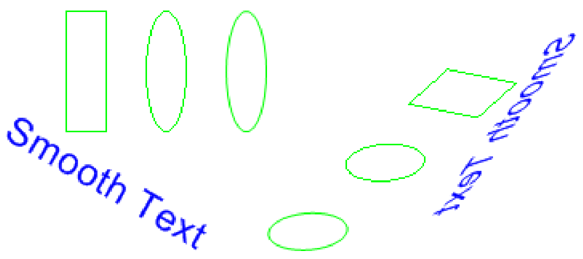
ORIGINAL POINT	DESTINATION POINT
Upper-left (0, 0)	(200, 20)
Upper-right (100, 0)	(110, 100)
Lower-left (0, 50)	(250, 30)

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).


(0, 0)
(100, 0)
x
(200, 20) is the destination for (0, 0).
(0, 50)
(250, 30) is the destination for (0, 50).
(110, 100) is the destination for (100, 0).
y

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.

```
    Point[] destinationPoints = {
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

    Image image = 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);
```

```
' 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

11/3/2020 • 3 minutes to read • Edit Online

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

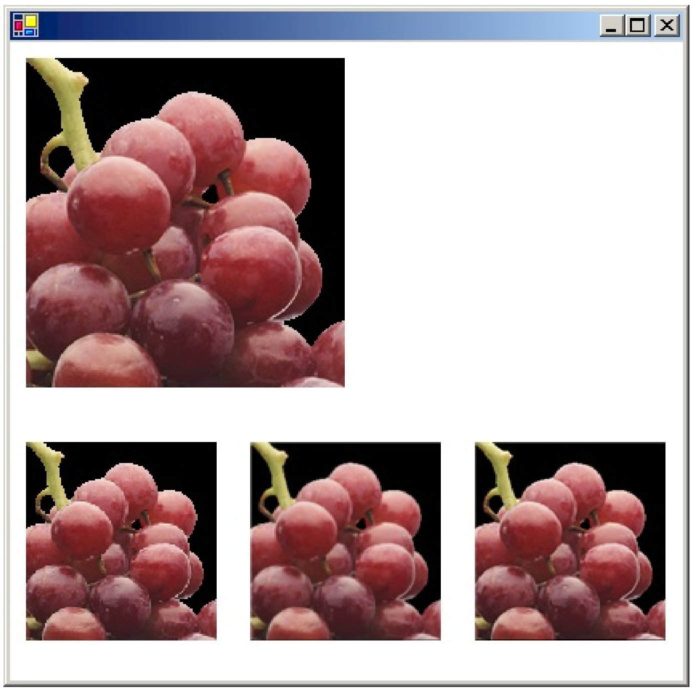
To stretch an image, each pixel in the original image must be mapped to a group of pixels in the larger image. To shrink an image, groups of pixels in the original image must be mapped to single pixels in the smaller image. The effectiveness of the algorithms that perform these mappings determines the 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.



```
Image image = new Bitmap("GrapeBunch.bmp");
int width = image.Width;
int height = image.Height;

// Draw the image with no shrinking or stretching.
e.Graphics.DrawImage(
    image,
    new Rectangle(10, 10, width, height),  // destination rectangle
    0,
    0,           // upper-left corner of source rectangle
    width,       // width of source rectangle
    height,      // height of source rectangle
    GraphicsUnit.Pixel,
    null);

// Shrink the image using low-quality interpolation.
e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor;
e.Graphics.DrawImage(
   image,
    new Rectangle(10, 250, (int)(0.6 * width), (int)(0.6 * height)),
    // destination rectangle
    0,
    0,           // upper-left corner of source rectangle
    width,       // width of source rectangle
    height,      // height of source rectangle
    GraphicsUnit.Pixel);

// Shrink the image using medium-quality interpolation.
e.Graphics.InterpolationMode = InterpolationMode.HighQualityBilinear;
e.Graphics.DrawImage(
    image,
    new Rectangle(150, 250, (int)(0.6 * width), (int)(0.6 * height)),
    // destination rectangle
    0,
    0,           // upper-left corner of source rectangle
    width,       // width of source rectangle
    height,      // height of source rectangle
    GraphicsUnit.Pixel);

// Shrink the image using high-quality interpolation.
e.Graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;
e.Graphics.DrawImage(
    image,
    new Rectangle(290, 250, (int)(0.6 * width), (int)(0.6 * height)),
    // destination rectangle
    0,
    0,           // upper-left corner of source rectangle
    width,       // width of source rectangle
    height,      // height of source rectangle
    GraphicsUnit.Pixel);
```

```
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

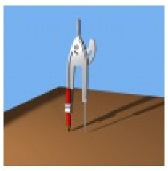
11/3/2020 • 2 minutes to read • Edit Online

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+.

```
public bool ThumbnailCallback()
{
    return true;
}

private void GetThumbnail(PaintEventArgs e)
{
    Image.GetThumbnailImageAbort callback =
        new Image.GetThumbnailImageAbort(ThumbnailCallback);
    Image image = new Bitmap(@"c:\FakePhoto.jpg");
    Image pThumbnail = image.GetThumbnailImage(100, 100, callback, new
       IntPtr());
    e.Graphics.DrawImage(
       pThumbnail,
       10,
       10,
       pThumbnail.Width,
       pThumbnail.Height);
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

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

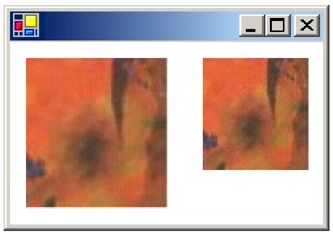
```
e.Graphics.DrawImage(image, 50, 30) ' upper-left corner at (50, 30)
```

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:



```
Image image = new Bitmap("Texture.jpg");

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

```
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

11/3/2020 • 4 minutes to read • Edit Online

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

NUMERIC VALUE	DESCRIPTION
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

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.

```
// Create an Image object.
Image image = new Bitmap(@"c:\FakePhoto.jpg");

// Get the PropertyItems property from image.
PropertyItem[] propItems = image.PropertyItems;

// Set up the display.
Font font = new Font("Arial", 12);
SolidBrush blackBrush = new SolidBrush(Color.Black);
int X = 0;
int Y = 0;

// For each PropertyItem in the array, display the ID, type, and
// length.
int count = 0;
foreach (PropertyItem 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++;
}
// Convert the value of the second property to a string, and display
// it.
System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
string manufacturer = encoding.GetString(propItems[1].Value);

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

```
'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)
```

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.
```

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

11/3/2020 • 2 minutes to read • Edit Online

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

Example

```
PictureBox pictureBox1 = new PictureBox();
public void CreateBitmapAtRuntime()
{
    pictureBox1.Size = new Size(210, 110);
    this.Controls.Add(pictureBox1);

    Bitmap flag = new Bitmap(200, 100);
    Graphics flagGraphics = Graphics.FromImage(flag);
    int red = 0;
    int white = 11;
    while (white <= 100) {
        flagGraphics.FillRectangle(Brushes.Red, 0, red, 200,10);
        flagGraphics.FillRectangle(Brushes.White, 0, white, 200, 10);
        red += 20;
        white += 20;
    }
    pictureBox1.Image = flag;
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
ListView listView1;
ImageList imageList1;

public void ExtractAssociatedIconEx()
{
    // Initialize the ListView, ImageList and Form.
    listView1 = new ListView();
    imageList1 = new ImageList();
    listView1.Location = new Point(37, 12);
    listView1.Size = new Size(151, 262);
    listView1.SmallImageList = imageList1;
    listView1.View = View.SmallIcon;
    this.ClientSize = new System.Drawing.Size(292, 266);
    this.Controls.Add(this.listView1);
    this.Text = "Form1";

    // Get the c:\ directory.
    System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(@"c:\");

    ListViewItem item;
    listView1.BeginUpdate();

    // For each file in the c:\ directory, create a ListViewItem
    // and set the icon to the icon extracted from the file.
    foreach (System.IO.FileInfo file in dir.GetFiles())
    {
        // Set a default icon for the file.
        Icon iconForFile = 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 (!imageList1.Images.ContainsKey(file.Extension))
        {
            // If not, add the image to the image list.
            iconForFile = System.Drawing.Icon.ExtractAssociatedIcon(file.FullName);
            imageList1.Images.Add(file.Extension, iconForFile);
        }
        item.ImageKey = file.Extension;
        listView1.Items.Add(item);
    }
    listView1.EndUpdate();
}
```

```
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

Alpha Blending Lines and Fills

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Draw Opaque and Semitransparent Lines

11/3/2020 • 2 minutes to read • Edit Online

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.

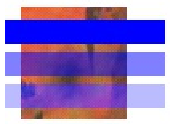
Example

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.

```
Bitmap bitmap = new Bitmap("Texture1.jpg");
e.Graphics.DrawImage(bitmap, 10, 5, bitmap.Width, bitmap.Height);

Pen opaquePen = new Pen(Color.FromArgb(255, 0, 0, 255), 15);
Pen semiTransPen = 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);
```

```
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)
```

The following illustration shows the output of the following code:



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

11/3/2020 • 2 minutes to read • Edit Online

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.

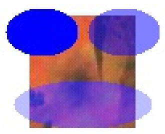
Example

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.

```
Bitmap bitmap = new Bitmap("Texture1.jpg");
e.Graphics.DrawImage(bitmap, 50, 50, bitmap.Width, bitmap.Height);

SolidBrush opaqueBrush = new SolidBrush(Color.FromArgb(255, 0, 0, 255));
SolidBrush semiTransBrush = 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);
```

```
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)
```

The following illustration shows the output of the following code:



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

11/3/2020 • 3 minutes to read • Edit Online

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.

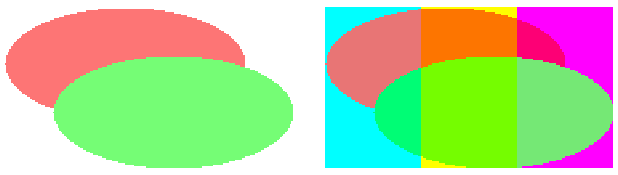
To create such a bitmap, construct a blank Bitmap object, and then construct a Graphics object based on that bitmap. Set the compositing mode of the Graphics object to CompositingMode.SourceCopy.

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:

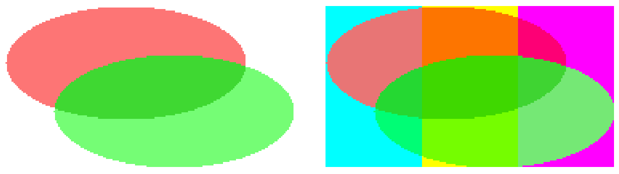
```
bitmapGraphics.CompositingMode = CompositingMode.SourceCopy;
```

```
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:

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

The following illustration shows the output of the revised code.



```
// Create a blank bitmap.
Bitmap myBitmap = new Bitmap(180, 100);

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

// Create a red brush and a green brush, each with an alpha value of 160.
SolidBrush redBrush = new SolidBrush(Color.FromArgb(160, 255, 0, 0));
SolidBrush greenBrush = 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.
SolidBrush colorBrush = 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);
```

```
' 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

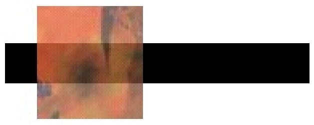
11/3/2020 • 3 minutes to read • Edit Online

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.

```
// Create the Bitmap object and load it with the texture image.
Bitmap bitmap = new Bitmap("Texture.jpg");

// Initialize the color matrix.
// Note the value 0.8 in row 4, column 4.
float[][] matrixItems ={
   new float[] {1, 0, 0, 0, 0},
   new float[] {0, 1, 0, 0, 0},
   new float[] {0, 0, 1, 0, 0},
   new float[] {0, 0, 0, 0.8f, 0},
   new float[] {0, 0, 0, 0, 1}};
ColorMatrix colorMatrix = new ColorMatrix(matrixItems);

// Create an ImageAttributes object and set its color matrix.
ImageAttributes imageAtt = 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.
int iWidth = bitmap.Width;
int iHeight = bitmap.Height;
e.Graphics.DrawImage(
   bitmap,
   new Rectangle(30, 0, iWidth, iHeight),  // destination rectangle
   0.0f,                          // source rectangle x
   0.0f,                          // source rectangle y
   iWidth,                        // source rectangle width
   iHeight,                       // source rectangle height
   GraphicsUnit.Pixel,
   imageAtt);
```

```
' 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

Using Fonts and Text

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Construct Font Families and Fonts

11/3/2020 • 2 minutes to read • Edit Online

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.

```
FontFamily fontFamily = new FontFamily("Arial");
Font font = new Font(
   fontFamily,
   16,
   FontStyle.Regular,
   GraphicsUnit.Pixel);
```

```
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

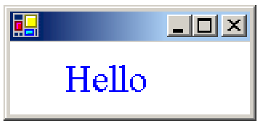
How to: Draw Text at a Specified Location

11/3/2020 • 2 minutes to read • Edit Online

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.

```
using (Font font1 = new Font("Times New Roman", 24, FontStyle.Bold, GraphicsUnit.Pixel)){
PointF pointF1 = new PointF(30, 10);
e.Graphics.DrawString("Hello", font1, Brushes.Blue, pointF1);
}
```

```
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.

```
using (Font font = new Font("Times New Roman", 24, FontStyle.Bold, GraphicsUnit.Pixel))
{
    Point point1 = new Point(30, 10);
    TextRenderer.DrawText(e.Graphics, "Hello", font, point1, Color.Blue);
}
```

```
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

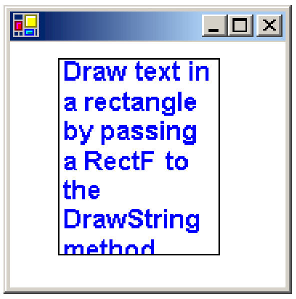
How to: Draw Wrapped Text in a Rectangle

11/3/2020 • 2 minutes to read • Edit Online

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.

```
string text1 = "Draw text in a rectangle by passing a RectF to the DrawString method.";
using (Font font1 = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point))
{
    RectangleF rectF1 = new RectangleF(30, 10, 100, 122);
    e.Graphics.DrawString(text1, font1, Brushes.Blue, rectF1);
    e.Graphics.DrawRectangle(Pens.Black, Rectangle.Round(rectF1));
}
```

```
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.

```
string text2 = "Draw text in a rectangle by passing a RectF to the DrawString method.";
using (Font font2 = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point))
{
    Rectangle rect2 = new Rectangle(30, 10, 100, 122);

    // Specify the text is wrapped.
    TextFormatFlags flags = TextFormatFlags.WordBreak;
    TextRenderer.DrawText(e.Graphics, text2, font2, rect2, Color.Blue, flags);
    e.Graphics.DrawRectangle(Pens.Black, Rectangle.Round(rect2));
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
private void RenderText6(PaintEventArgs e)
{
    TextFormatFlags flags = TextFormatFlags.Bottom | TextFormatFlags.EndEllipsis;
    TextRenderer.DrawText(e.Graphics, "This is some text that will be clipped at the end.", this.Font,
        new Rectangle(10, 10, 100, 50), SystemColors.ControlText, flags);
}
```

```
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

- Using Fonts and Text

How to: Align Drawn Text

11/3/2020 • 2 minutes to read • Edit Online

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.

```
string text1 = "Use StringFormat and Rectangle objects to"
    + " center text in a rectangle.";
using (Font font1 = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point))
{
    Rectangle rect1 = 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.
    StringFormat stringFormat = 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);
}
```

```
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.

```
string text2 = "Use TextFormatFlags and Rectangle objects to"
 + " center text in a rectangle.";

using (Font font2 = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point))
{
    Rectangle rect2 = new Rectangle(150, 10, 130, 140);

    // Create a TextFormatFlags with word wrapping, horizontal center and
    // vertical center specified.
    TextFormatFlags flags = TextFormatFlags.HorizontalCenter |
        TextFormatFlags.VerticalCenter | 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);
}
```

```
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

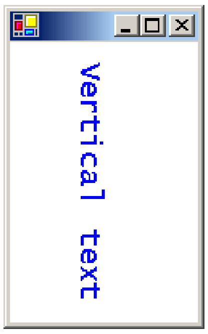
11/3/2020 • 2 minutes to read • Edit Online

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:



```
string myText = "Vertical text";

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

stringFormat.FormatFlags = StringFormatFlags.DirectionVertical;

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

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

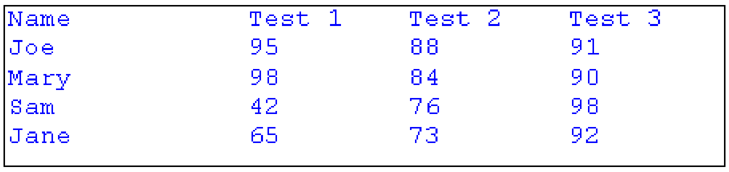
NOTE

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:



```
Name            Test 1     Test 2     Test 3
Joe             95         88         91
Mary            98         84         90
Sam             42         76         98
Jane            65         73         92
```

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.

```
string text = "Name\tTest 1\tTest 2\tTest 3\n";
text = text + "Joe\t95\t88\t91\n";
text = text + "Mary\t98\t84\t90\n";
text = text + "Sam\t42\t76\t98\n";
text = text + "Jane\t65\t73\t92\n";

FontFamily fontFamily = new FontFamily("Courier New");
Font font = new Font(
   fontFamily,
   12,
   FontStyle.Regular,
   GraphicsUnit.Point);
Rectangle rect = new Rectangle(10, 10, 450, 100);
StringFormat stringFormat = new StringFormat();
SolidBrush solidBrush = new SolidBrush(Color.FromArgb(255, 0, 0, 255));
float[] tabs = { 150, 100, 100, 100 };

stringFormat.SetTabStops(0, tabs);

e.Graphics.DrawString(text, font, solidBrush, rect, stringFormat);

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

```
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

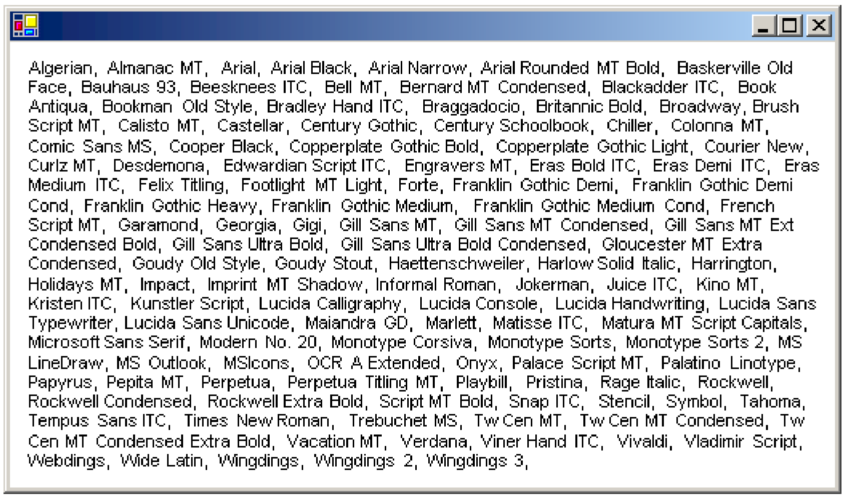
11/3/2020 • 2 minutes to read • Edit Online

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:



```
Algerian, Almanac MT, Arial, Arial Black, Arial Narrow, Arial Rounded MT Bold, Baskerville Old
Face, Bauhaus 93, Beesknees ITC, Bell MT, Bernard MT Condensed, Blackadder ITC, Book
Antiqua, Bookman Old Style, Bradley Hand ITC, Braggadocio, Britannic Bold, Broadway, Brush
Script MT, Calisto MT, Castellar, Century Gothic, Century Schoolbook, Chiller, Colonna MT,
Comic Sans MS, Cooper Black, Copperplate Gothic Bold, Copperplate Gothic Light, Courier New,
Curlz MT, Desdemona, Edwardian Script ITC, Engravers MT, Eras Bold ITC, Eras Demi ITC, Eras
Medium ITC, Felix Titling, Footlight MT Light, Forte, Franklin Gothic Demi, Franklin Gothic Demi
Cond, Franklin Gothic Heavy, Franklin Gothic Medium, Franklin Gothic Medium Cond, French
Script MT, Garamond, Georgia, Gigi, Gill Sans MT, Gill Sans MT Condensed, Gill Sans MT Ext
Condensed Bold, Gill Sans Ultra Bold, Gill Sans Ultra Bold Condensed, Gloucester MT Extra
Condensed, Goudy Old Style, Goudy Stout, Haettenschweiler, Harlow Solid Italic, Harrington,
Holidays MT, Impact, Imprint MT Shadow, Informal Roman, Jokerman, Juice ITC, Kino MT,
Kristen ITC, Kunstler Script, Lucida Calligraphy, Lucida Console, Lucida Handwriting, Lucida Sans
Typewriter, Lucida Sans Unicode, Maiandra GD, Marlett, Matisse ITC, Matura MT Script Capitals,
Microsoft Sans Serif, Modern No. 20, Monotype Corsiva, Monotype Sorts, Monotype Sorts 2, MS
LineDraw, MS Outlook, MSIcons, OCR A Extended, Onyx, Palace Script MT, Palatino Linotype,
Papyrus, Pepita MT, Perpetua, Perpetua Titling MT, Playbill, Pristina, Rage Italic, Rockwell,
Rockwell Condensed, Rockwell Extra Bold, Script MT Bold, Snap ITC, Stencil, Symbol, Tahoma,
Tempus Sans ITC, Times New Roman, Trebuchet MS, Tw Cen MT, Tw Cen MT Condensed, Tw
Cen MT Condensed Extra Bold, Vacation MT, Verdana, Viner Hand ITC, Vivaldi, Vladimir Script,
Webdings, Wide Latin, Wingdings, Wingdings 2, Wingdings 3,
```

```csharp
FontFamily fontFamily = new FontFamily("Arial");
Font font = new Font(
   fontFamily,
   8,
   FontStyle.Regular,
   GraphicsUnit.Point);
RectangleF rectF = new RectangleF(10, 10, 500, 500);
SolidBrush solidBrush = new SolidBrush(Color.Black);

string familyName;
string familyList = "";
FontFamily[] fontFamilies;

InstalledFontCollection installedFontCollection = 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.

int count = fontFamilies.Length;
for (int j = 0; j < count; ++j)
{
    familyName = fontFamilies[j].Name;
    familyList = familyList + familyName;
    familyList = familyList + ",  ";
}

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

```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

11/3/2020 • 6 minutes to read • Edit Online

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:

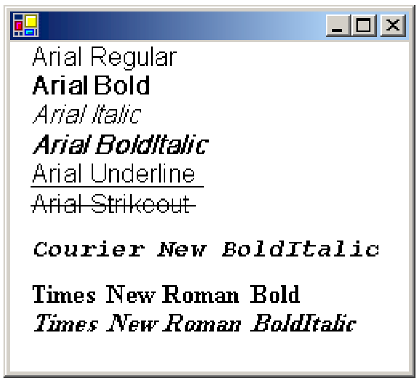
- C:\systemroot\Fonts\Arial.tff (Arial, regular)
- C:\systemroot\Fonts\CourBI.tff (Courier New, bold italic)
- C:\systemroot\Fonts\TimesBd.tff (Times New Roman, bold)

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 Regular
Arial Bold
Arial Italic
Arial BoldItalic
Arial Underline
Arial Strikeout

Courier New BoldItalic

Times New Roman Bold
Times New Roman BoldItalic
```

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.

```
PointF pointF = new PointF(10, 0);
SolidBrush solidBrush = new SolidBrush(Color.Black);

int count = 0;
string familyName = "";
string familyNameAndStyle;
FontFamily[] fontFamilies;
PrivateFontCollection privateFontCollection = 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.
for (int j = 0; j < count; ++j)
{
    // Get the font family name.
    familyName = fontFamilies[j].Name;

    // Is the regular style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Regular))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + " Regular";

        Font regFont = new Font(
           familyName,
           16,
           FontStyle.Regular,
           GraphicsUnit.Pixel);

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

        pointF.Y += regFont.Height;
    }

    // Is the bold style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Bold))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + " Bold";

        Font boldFont = new Font(
           familyName,
           16,
           FontStyle.Bold,
           GraphicsUnit.Pixel);

        e.Graphics.DrawString(familyNameAndStyle, boldFont, solidBrush, pointF);
        pointF.Y += boldFont.Height;
    }
    // Is the italic style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Italic))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + " Italic";

        Font italicFont = new Font(
           familyName,
           16,
           FontStyle.Italic,
           GraphicsUnit.Pixel);

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

        pointF.Y += italicFont.Height;
    }

    // Is the bold italic style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Italic) &&
    fontFamilies[j].IsStyleAvailable(FontStyle.Bold))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + "BoldItalic";

        Font italicFont = new Font(
           familyName,
           16,
           FontStyle.Italic | FontStyle.Bold,
           GraphicsUnit.Pixel);

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

        pointF.Y += italicFont.Height;
    }
    // Is the underline style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Underline))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + " Underline";

        Font underlineFont = new Font(
           familyName,
           16,
           FontStyle.Underline,
           GraphicsUnit.Pixel);

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

        pointF.Y += underlineFont.Height;
    }
    // Is the strikeout style available?
    if (fontFamilies[j].IsStyleAvailable(FontStyle.Strikeout))
    {
        familyNameAndStyle = "";
        familyNameAndStyle = familyNameAndStyle + familyName;
        familyNameAndStyle = familyNameAndStyle + " Strikeout";

        Font strikeFont = new Font(
           familyName,
           16,
           FontStyle.Strikeout,
           GraphicsUnit.Pixel);

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

        pointF.Y += strikeFont.Height;
    }

    // Separate the families with white space.
    pointF.Y += 10;
} // for
```

```
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

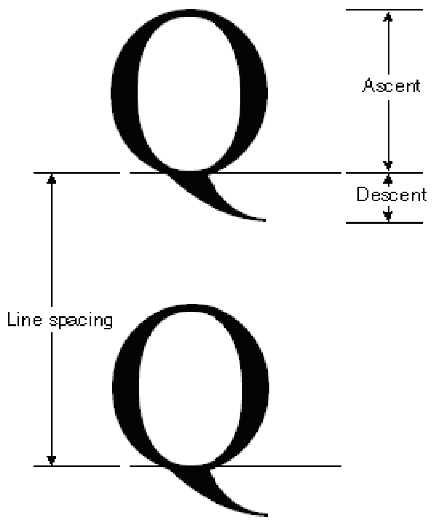
11/3/2020 • 5 minutes to read • Edit Online

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

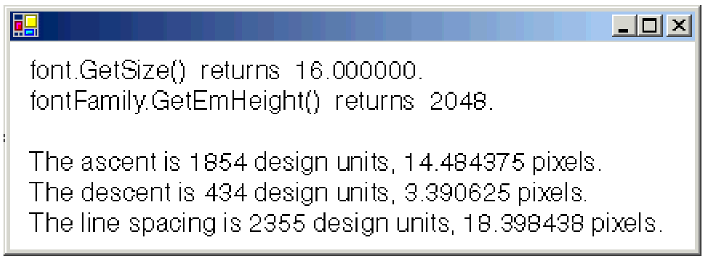
The following illustration shows the various metrics:



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:



```
font.GetSize() returns 16.000000.
fontFamily.GetEmHeight() returns 2048.

The ascent is 1854 design units, 14.484375 pixels.
The descent is 434 design units, 3.390625 pixels.
The line spacing is 2355 design units, 18.398438 pixels.
```

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:

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

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.

```
string infoString = "";  // enough space for one line of output
int ascent;             // font family ascent in design units
float ascentPixel;      // ascent converted to pixels
int descent;            // font family descent in design units
float descentPixel;     // descent converted to pixels
int lineSpacing;        // font family line spacing in design units
float lineSpacingPixel; // line spacing converted to pixels

FontFamily fontFamily = new FontFamily("Arial");
Font font = new Font(
   fontFamily,
   16, FontStyle.Regular,
   GraphicsUnit.Pixel);
PointF pointF = new PointF(10, 10);
SolidBrush solidBrush = new SolidBrush(Color.Black);

// Display the font size in pixels.
infoString = "font.Size returns " + font.Size + ".";
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);
```

```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

11/3/2020 • 2 minutes to read • Edit Online

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

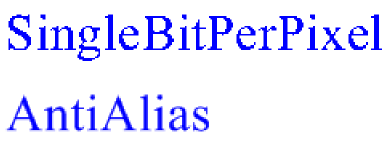
Example

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

```
FontFamily fontFamily = new FontFamily("Times New Roman");
Font font = new Font(
   fontFamily,
   32,
   FontStyle.Regular,
   GraphicsUnit.Pixel);
SolidBrush solidBrush = new SolidBrush(Color.FromArgb(255, 0, 0, 255));
string string1 = "SingleBitPerPixel";
string string2 = "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));
```

```
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))
```

The following illustration shows the output of the example code:



SingleBitPerPixel

AntiAlias

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

Constructing and Drawing Curves

11/3/2020 • 2 minutes to read • Edit Online

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.

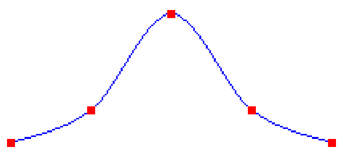
How to: Draw Cardinal Splines

11/3/2020 • 2 minutes to read • Edit Online

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.

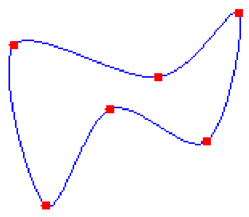


```
Point[] points = {
   new Point(0, 100),
   new Point(50, 80),
   new Point(100, 20),
   new Point(150, 80),
   new Point(200, 100)};

Pen pen = new Pen(Color.FromArgb(255, 0, 0, 255));
e.Graphics.DrawCurve(pen, points);
```

```
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:

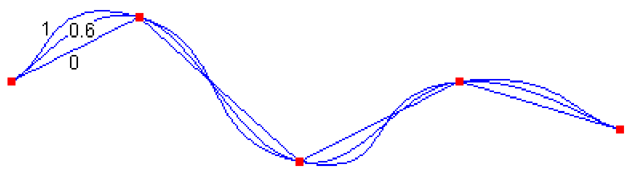


```
Point[] points = {
   new Point(60, 60),
   new Point(150, 80),
   new Point(200, 40),
   new Point(180, 120),
   new Point(120, 100),
   new Point(80, 160)};

Pen pen = new Pen(Color.FromArgb(255, 0, 0, 255));
e.Graphics.DrawClosedCurve(pen, points);
```

```
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.



```
Point[] points = {
   new Point(20, 50),
   new Point(100, 10),
   new Point(200, 100),
   new Point(300, 50),
   new Point(400, 80)};

Pen pen = 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);
```

```
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

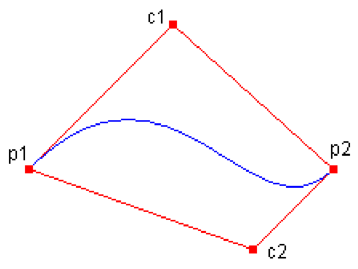
11/3/2020 • 2 minutes to read • Edit Online

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.



```
Point p1 = new Point(10, 100);   // Start point
Point c1 = new Point(100, 10);   // First control point
Point c2 = new Point(150, 150);  // Second control point
Point p2 = new Point(200, 100);  // Endpoint

Pen pen = new Pen(Color.FromArgb(255, 0, 0, 255));
e.Graphics.DrawBezier(pen, p1, c1, c2, p2);
```

```
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

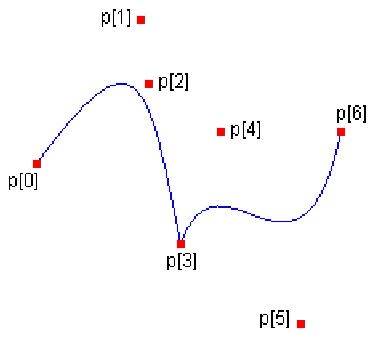
11/3/2020 • 2 minutes to read • Edit Online

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:



```
Point[] p = {
   new Point(10, 100),   // start point of first spline
   new Point(75, 10),    // first control point of first spline
   new Point(80, 50),    // second control point of first spline

   new Point(100, 150),  // endpoint of first spline and
                         // start point of second spline

   new Point(125, 80),   // first control point of second spline
   new Point(175, 200),  // second control point of second spline
   new Point(200, 80)};  // endpoint of second spline

Pen pen = new Pen(Color.Blue);
e.Graphics.DrawBeziers(pen, p);
```

```
' 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

Constructing and Drawing Paths

11/3/2020 • 2 minutes to read • Edit Online

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.

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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
GraphicsPath path = new GraphicsPath();
path.AddArc(175, 50, 50, 50, 0, -180);
e.Graphics.DrawPath(new Pen(Color.FromArgb(128, 255, 0, 0), 4), path);
```

```
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)
```

```
     // Create an array of points for the curve in the second figure.
     Point[] points = {
new Point(40, 60),
new Point(50, 70),
new Point(30, 90)};

     GraphicsPath path = 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);
```

```
' 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

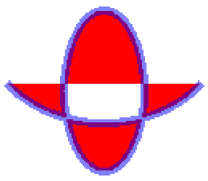
11/3/2020 • 2 minutes to read • Edit Online

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.



```
GraphicsPath path = 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);

Pen pen = new Pen(Color.FromArgb(128, 0, 0, 255), 5);
SolidBrush brush = new SolidBrush(Color.Red);

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

```
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.

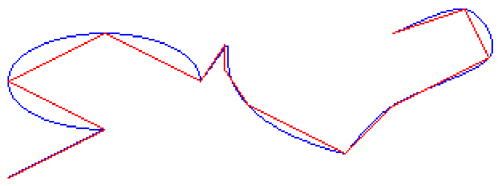
See also

- GraphicsPath
- Graphics Paths in GDI+

How to: Flatten a Curved Path into a Line

11/3/2020 • 2 minutes to read • Edit Online

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 Transformations in Managed GDI+

11/3/2020 • 2 minutes to read • Edit Online

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 the World Transformation

11/3/2020 • 2 minutes to read • Edit Online

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.

```
Rectangle rect = new Rectangle(0, 0, 50, 50);
Pen pen = new Pen(Color.FromArgb(128, 200, 0, 200), 2);
e.Graphics.DrawRectangle(pen, rect);
```

```
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:

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

```
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:

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

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

To translate the rectangle, use the following code:

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

```
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+

Why Transformation Order Is Significant

11/3/2020 • 3 minutes to read • Edit Online

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.

```
Rectangle rect = new Rectangle(0, 0, 50, 50);
Pen pen = 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);
```

```
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.

```
Rectangle rect = new Rectangle(0, 0, 50, 50);
Pen pen = 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);
```

```
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.

```
Rectangle rect = new Rectangle(0, 0, 50, 50);
Pen pen = 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);
```

```
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 Graphics Containers

11/3/2020 • 2 minutes to read • Edit Online

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.

Managing the State of a Graphics Object

11/3/2020 • 3 minutes to read • Edit Online

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.

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

```
Graphics graphics = e.Graphics;
Pen pen = 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:

```
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)
```

```
Graphics graphics = e.Graphics;
Pen pen = 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

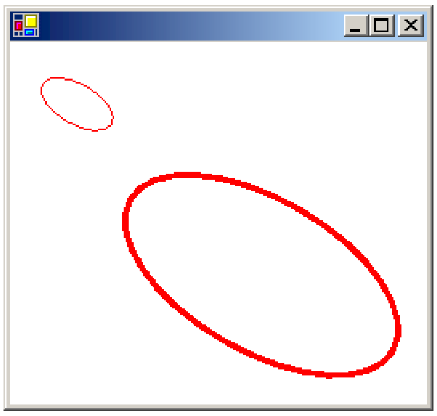
A Graphics object maintains two transformations (world and page) that are applied to all items drawn by that Graphics object. Any affine transformation can be stored in the world transformation. Affine transformations include scaling, rotating, reflecting, skewing, and translating. The page transformation 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.

```
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)
```

```
Graphics graphics = e.Graphics;
Pen pen = 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

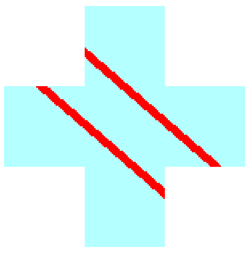
A Graphics object maintains a clipping region that applies to all items drawn by that Graphics object. You can set the clipping region by calling the SetClip method.

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.

```
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)
```

```
Graphics graphics = e.Graphics;

// Opaque red, width 5
Pen pen = new Pen(Color.Red, 5);

// Opaque aqua
SolidBrush brush = new SolidBrush(Color.FromArgb(255, 180, 255, 255));

// Create a plus-shaped region by forming the union of two rectangles.
Region region = 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

Using Nested Graphics Containers

11/3/2020 • 5 minutes to read • Edit Online

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.

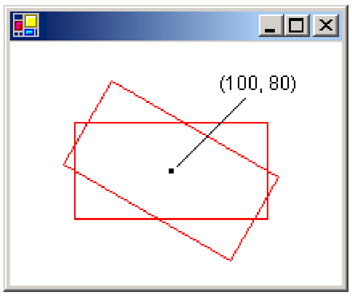
Transformations in Nested Containers

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.

```
Graphics graphics = e.Graphics;
Pen pen = new Pen(Color.Red);
GraphicsContainer 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);
```

```
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:



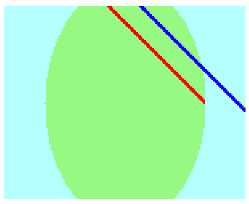
Clipping in Nested Containers

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.

```
Graphics graphics = e.Graphics;
GraphicsContainer graphicsContainer;
Pen redPen = new Pen(Color.Red, 2);
Pen bluePen = new Pen(Color.Blue, 2);
SolidBrush aquaBrush = new SolidBrush(Color.FromArgb(255, 180, 255, 255));
SolidBrush greenBrush = 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.
GraphicsPath path = new GraphicsPath();
path.AddEllipse(75, 50, 100, 150);

// Construct a region based on the path.
Region region = 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);
```

```
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

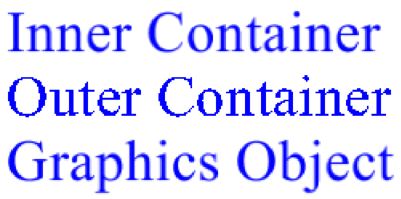
You are not limited to one container in a Graphics object. You can create a sequence of containers, each nested in the preceding, and you can specify the world transformation, clipping region, and quality settings of each of those nested containers. If you call a drawing method from inside the innermost 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.

```
Graphics graphics = e.Graphics;
GraphicsContainer innerContainer;
GraphicsContainer outerContainer;
SolidBrush brush = new SolidBrush(Color.Blue);
FontFamily fontFamily = new FontFamily("Times New Roman");
Font font = 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));
```

```
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

Using Regions

11/3/2020 • 2 minutes to read • Edit Online

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.

How to: Use Hit Testing with a Region

11/3/2020 • 2 minutes to read • Edit Online

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.

```
Point point = 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.

SolidBrush solidBrush = new SolidBrush(Color.Black);
Region region1 = new Region(new Rectangle(50, 0, 50, 150));
Region region2 = 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))
{
    // 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);
}

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

```
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

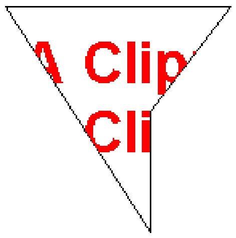
11/3/2020 • 2 minutes to read • Edit Online

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:



```
     // Create a path that consists of a single polygon.
     Point[] polyPoints = {
new Point(10, 10),
new Point(150, 10),
new Point(100, 75),
new Point(100, 150)};
     GraphicsPath path = new GraphicsPath();
     path.AddPolygon(polyPoints);

     // Construct a region based on the path.
     Region region = new Region(path);

     // Draw the outline of the region.
     Pen 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.
     FontFamily fontFamily = new FontFamily("Arial");
     Font font = new Font(
        fontFamily,
        36, FontStyle.Bold,
        GraphicsUnit.Pixel);
     SolidBrush solidBrush = 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));
```

```
' 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

Recoloring Images

11/3/2020 • 2 minutes to read • Edit Online

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.

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

11/3/2020 • 3 minutes to read • Edit Online

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.

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

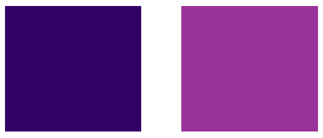
Identity Matrix

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.

```
Image image = new Bitmap("InputColor.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;

float[][] colorMatrixElements = {
   new float[] {2,  0,  0,  0, 0},        // red scaling factor of 2
   new float[] {0,  1,  0,  0, 0},        // green scaling factor of 1
   new float[] {0,  0,  1,  0, 0},        // blue scaling factor of 1
   new float[] {0,  0,  0,  1, 0},        // alpha scaling factor of 1
   new float[] {.2f, .2f, .2f, 0, 1}};    // three translations of 0.2

ColorMatrix colorMatrix = 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),  // destination rectangle
   0, 0,        // upper-left corner of source rectangle
   width,       // width of source rectangle
   height,      // height of source rectangle
   GraphicsUnit.Pixel,
   imageAttributes);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

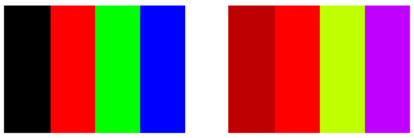
A translation adds a value to one or more of the four color components. The color matrix entries that represent translations are given in the following table.

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)

```csharp
Image image = new Bitmap("ColorBars.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;

float[][] colorMatrixElements = {
   new float[] {1,  0,  0,  0, 0},
   new float[] {0,  1,  0,  0, 0},
   new float[] {0,  0,  1,  0, 0},
   new float[] {0,  0,  0,  1, 0},
   new float[] {.75f, 0, 0, 0, 1}};

ColorMatrix colorMatrix = 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),  // destination rectangle
   0, 0,        // upper-left corner of source rectangle
   width,       // width of source rectangle
   height,      // height of source rectangle
   GraphicsUnit.Pixel,
   imageAttributes);
```

```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

11/3/2020 • 4 minutes to read • Edit Online

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]

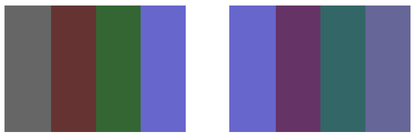
Scaling One Color

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.

```
Image image = new Bitmap("ColorBars2.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;

float[][] colorMatrixElements = {
   new float[] {1,  0,  0,  0, 0},
   new float[] {0,  1,  0,  0, 0},
   new float[] {0,  0,  2,  0, 0},
   new float[] {0,  0,  0,  1, 0},
   new float[] {0, 0, 0, 0, 1}};

ColorMatrix colorMatrix = 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),  // destination rectangle
   0, 0,        // upper-left corner of source rectangle
   width,       // width of source rectangle
   height,      // height of source rectangle
   GraphicsUnit.Pixel,
   imageAttributes);
```

```
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)

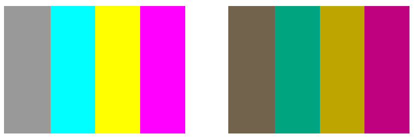
Scaling Multiple Colors

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.

```
Image image = new Bitmap("ColorBars.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;

float[][] colorMatrixElements = {
   new float[] {.75F,  0,  0,  0, 0},
   new float[] {0,  .65F,  0,  0, 0},
   new float[] {0,  0,  .5F,  0, 0},
   new float[] {0,  0,  0,  1F, 0},
   new float[] {0, 0, 0, 0, 1F}};

ColorMatrix colorMatrix = 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),  // destination rectangle
   0, 0,        // upper-left corner of source rectangle
   width,       // width of source rectangle
   height,      // height of source rectangle
   GraphicsUnit.Pixel,
   imageAttributes);
```

```
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)

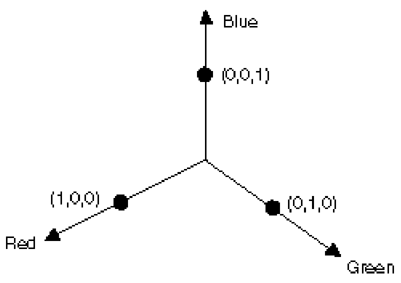
See also

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

How to: Rotate Colors

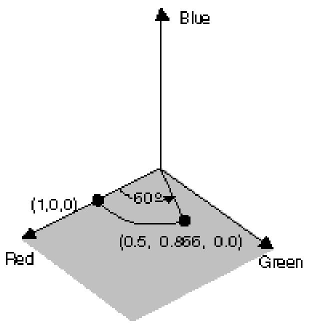
11/3/2020 • 2 minutes to read • Edit Online

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.

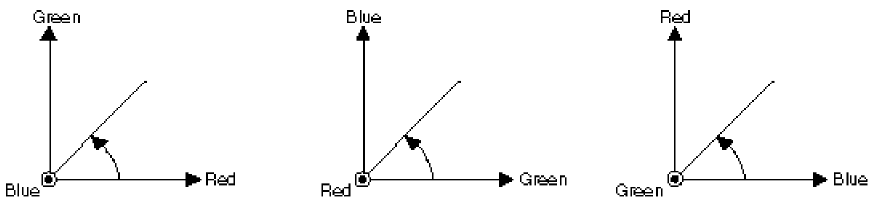


A color can be thought of as a point in 3D space. For example, the point (1, 0, 0) in space represents the color red, and the point (0, 1, 0) in space represents the color green.

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):



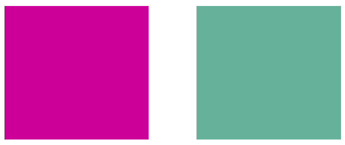
$$\begin{bmatrix} \cos & \sin & 0 & 0 & 0 \\ -\sin & \cos & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos & \sin & 0 & 0 \\ 0 & -\sin & \cos & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos & 0 & -\sin & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \sin & 0 & \cos & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

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

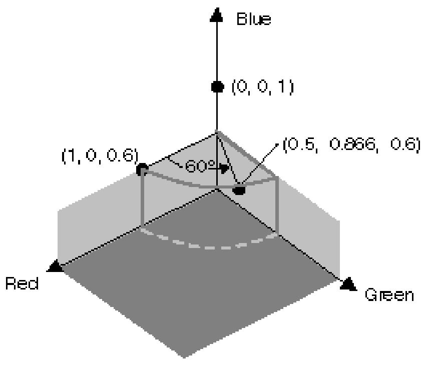
Example

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.

```
private void RotateColors(PaintEventArgs e)
{
    Bitmap image = new Bitmap("RotationInput.bmp");
    ImageAttributes imageAttributes = new ImageAttributes();
    int width = image.Width;
    int height = image.Height;
    float degrees = 60f;
    double r = degrees * System.Math.PI / 180; // degrees to radians

    float[][] colorMatrixElements = {
        new float[] {(float)System.Math.Cos(r),  (float)System.Math.Sin(r),  0,  0, 0},
        new float[] {(float)-System.Math.Sin(r),  (float)-System.Math.Cos(r),  0,  0, 0},
        new float[] {0,  0,  2,  0, 0},
        new float[] {0,  0,  0,  1, 0},
        new float[] {0, 0, 0, 0, 1}};

    ColorMatrix colorMatrix = 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),  // destination rectangle
        0, 0,        // upper-left corner of source rectangle
        width,       // width of source rectangle
        height,      // height of source rectangle
        GraphicsUnit.Pixel,
       imageAttributes);
}
```

```
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

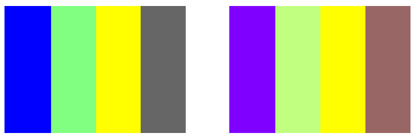
11/3/2020 • 2 minutes to read • Edit Online

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)

```
Image image = new Bitmap("ColorBars.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;

float[][] colorMatrixElements = {
        new float[] {1,  0,  0,  0, 0},
        new float[] {0,  1,  0,  0, 0},
        new float[] {0.5f,  0,  1,  0, 0},
        new float[] {0,  0,  0,  1, 0},
        new float[] {0, 0, 0, 0, 1}};

ColorMatrix colorMatrix = 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),  // destination rectangle
    0, 0,        // upper-left corner of source rectangle
    width,       // width of source rectangle
    height,      // height of source rectangle
    GraphicsUnit.Pixel,
   imageAttributes);
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

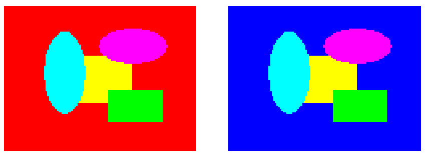
When GDI+ draws an image, each pixel of the image is compared to the array of old colors. If a pixel's color matches an old color, its color is changed to the corresponding new color. The colors are changed only for rendering — the color values of the image itself (stored in an Image or Bitmap object) 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.



```
Image image = new Bitmap("RemapInput.bmp");
ImageAttributes imageAttributes = new ImageAttributes();
int width = image.Width;
int height = image.Height;
ColorMap colorMap = new ColorMap();

colorMap.OldColor = Color.FromArgb(255, 255, 0, 0);  // opaque red
colorMap.NewColor = Color.FromArgb(255, 0, 0, 255);  // opaque blue

ColorMap[] remapTable = { colorMap };

imageAttributes.SetRemapTable(remapTable, ColorAdjustType.Bitmap);

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

e.Graphics.DrawImage(
   image,
   new Rectangle(150, 10, width, height),  // destination rectangle
   0, 0,        // upper-left corner of source rectangle
   width,       // width of source rectangle
   height,      // height of source rectangle
   GraphicsUnit.Pixel,
   imageAttributes);
```

```
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

Using Image Encoders and Decoders in Managed GDI+

11/3/2020 • 2 minutes to read • Edit Online

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

How to: List Installed Encoders

11/3/2020 • 3 minutes to read • Edit Online

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.

```
private void GetImageEncodersExample(PaintEventArgs e)
{
    // Get an array of available encoders.
    ImageCodecInfo[] myCodecs;
    myCodecs = ImageCodecInfo.GetImageEncoders();
    int numCodecs = myCodecs.GetLength(0);

    // Set up display variables.
    Color foreColor = Color.Black;
    Font font = new Font("Arial", 8);
    int i = 0;

    // Check to determine whether any codecs were found.
    if (numCodecs > 0)
    {
        // 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.
        string[] myCodecInfo = new string[numCodecs * 10];

        // Write all the codec information to the array.
        for (i = 0; i < numCodecs; i++)
        {
            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] = " ";
        }
        int numMyCodecInfo = myCodecInfo.GetLength(0);

        // Render all of the information to the screen.
        int j = 20;
        for (i = 0; i < numMyCodecInfo; i++)
        {
            e.Graphics.DrawString(myCodecInfo[i],
                font,
                new SolidBrush(foreColor),
                20,
                j);
            j += 12;
        }
    }
    else
        e.Graphics.DrawString("No Codecs Found",
            font,
            new SolidBrush(foreColor),
            20,
            20);
}
```

```
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

11/3/2020 • 3 minutes to read • Edit Online

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.

```
private void GetImageDecodersExample(PaintEventArgs e)
{
    // Get an array of available decoders.
    ImageCodecInfo[] myCodecs;
    myCodecs = ImageCodecInfo.GetImageDecoders();
    int numCodecs = myCodecs.GetLength(0);

    // Set up display variables.
    Color foreColor = Color.Black;
    Font font = new Font("Arial", 8);
    int i = 0;

    // Check to determine whether any codecs were found.
    if (numCodecs > 0)
    {
        // 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.
        string[] myCodecInfo = new string[numCodecs * 10];

        // Write all the codec information to the array.
        for (i = 0; i < numCodecs; i++)
        {
            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] = " ";
        }
        int numMyCodecInfo = myCodecInfo.GetLength(0);

        // Render all of the information to the screen.
        int j = 20;
        for (i = 0; i < numMyCodecInfo; i++)
        {
            e.Graphics.DrawString(myCodecInfo[i],
                font,
                new SolidBrush(foreColor),
                20,
                j);
            j += 12;
        }
    }
    else
        e.Graphics.DrawString("No Codecs Found",
            font,
            new SolidBrush(foreColor),
            20,
            20);
}
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
private void GetSupportedParameters(PaintEventArgs e)
{
    Bitmap bitmap1 = new Bitmap(1, 1);
    ImageCodecInfo jpgEncoder = GetEncoder(ImageFormat.Jpeg);
    EncoderParameters paramList = bitmap1.GetEncoderParameterList(jpgEncoder.Clsid);
    EncoderParameter[] encParams = paramList.Param;
    StringBuilder paramInfo = new StringBuilder();

    for (int i = 0; i < encParams.Length; i++)
    {
        paramInfo.Append("Param " + i + " holds " + encParams[i].NumberOfValues +
            " items of type " +
            encParams[i].ValueType + "\r\n" + "Guid category: " + encParams[i].Encoder.Guid + "\r\n");
    }
    e.Graphics.DrawString(paramInfo.ToString(), this.Font, Brushes.Red, 10.0F, 10.0F);
}

private ImageCodecInfo GetEncoder(ImageFormat format)
{
    ImageCodecInfo[] codecs = ImageCodecInfo.GetImageEncoders();

    foreach (ImageCodecInfo codec in codecs)
    {
        if (codec.FormatID == format.Guid)
        {
            return codec;
        }
    }

    return null;
}
```

```
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.GetImageEncoders()

    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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
private void SaveBmpAsPNG()
{
    Bitmap bmp1 = new Bitmap(typeof(Button), "Button.bmp");
    bmp1.Save(@"c:\button.png", ImageFormat.Png);
}
```

```
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

3/9/2021 • 2 minutes to read • Edit Online

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.

```
private void VaryQualityLevel()
    {
        // Get a bitmap. The using statement ensures objects
        // are automatically disposed from memory after use.
        using (Bitmap bmp1 = new Bitmap(@"C:\TestPhoto.jpg"))
        {
            ImageCodecInfo jpgEncoder = GetEncoder(ImageFormat.Jpeg);

            // Create an Encoder object based on the GUID
            // for the Quality parameter category.
            System.Drawing.Imaging.Encoder myEncoder =
                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.
            EncoderParameters myEncoderParameters = new EncoderParameters(1);

            EncoderParameter myEncoderParameter = new EncoderParameter(myEncoder, 50L);
            myEncoderParameters.Param[0] = myEncoderParameter;
            bmp1.Save(@"c:\TestPhotoQualityFifty.jpg", jpgEncoder, myEncoderParameters);

            myEncoderParameter = new EncoderParameter(myEncoder, 100L);
            myEncoderParameters.Param[0] = myEncoderParameter;
            bmp1.Save(@"C:\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:\TestPhotoQualityZero.jpg", jpgEncoder, myEncoderParameters);
        }
    }
```

```
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
```

```
private ImageCodecInfo GetEncoder(ImageFormat format)
{
    ImageCodecInfo[] codecs = ImageCodecInfo.GetImageEncoders();
    foreach (ImageCodecInfo codec in codecs)
    {
        if (codec.FormatID == format.Guid)
        {
            return codec;
        }
    }
    return null;
}
```

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

    Dim codecs As ImageCodecInfo() = ImageCodecInfo.GetImageEncoders()
    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+

Using Double Buffering

11/3/2020 • 2 minutes to read • Edit Online

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.

Double Buffered Graphics

11/3/2020 • 2 minutes to read • Edit Online

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 Current. 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, 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 Graphics, 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 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

11/3/2020 • 2 minutes to read • Edit Online

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`.

```
DoubleBuffered = true;
```

```
DoubleBuffered = True
```

- or -

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

```
SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
```

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

See also

- DoubleBuffered
- SetStyle
- Double Buffered Graphics
- Graphics and Drawing in Windows Forms

How to: Manually Manage Buffered Graphics

11/3/2020 • 2 minutes to read • Edit Online

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.

```
BufferedGraphicsContext myContext;
myContext = BufferedGraphicsManager.Current;
```

```
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.

```
BufferedGraphicsContext myContext;
myContext = new BufferedGraphicsContext();
// Insert code to create graphics here.
// On a non-default BufferedGraphicsContext instance, you should always
// call Dispose when finished.
myContext.Dispose();
```

```
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

11/3/2020 • 2 minutes to read • Edit Online

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.

```
// This example assumes the existence of a form called Form1.
BufferedGraphicsContext currentContext;
BufferedGraphics myBuffer;
// 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(this.CreateGraphics(),
   this.DisplayRectangle);
```

```
' 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:

```
// Draws an ellipse to the graphics buffer.
myBuffer.Graphics.DrawEllipse(Pens.Blue, this.DisplayRectangle);
```

```
' 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.

```
// This example assumes the existence of a BufferedGraphics instance
// called myBuffer.
// 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(this.CreateGraphics());
```

```
' 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.

```
myBuffer.Dispose();
```

```
myBuffer.Dispose()
```

See also

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

Application Settings for Windows Forms

3/9/2021 • 2 minutes to read • Edit Online

The Applications Settings feature of Windows Forms makes it easy to create, store, and maintain custom application and user preferences on the client. With Application Settings, you can store not only application data such as database connection strings, but also user-specific data, such as toolbar positions and most-recently used lists.

In This Section

Application Settings Overview
Discusses how to create and store settings data on behalf of your application and your users.

Application Settings Architecture
Describes how the Application Settings feature works, and explores advanced features of the architecture such as grouped settings and settings keys.

Application Settings Attributes
Lists and describes the attributes that can be applied to an application settings wrapper class or its settings properties.

Application Settings for Custom Controls
Discusses what must be done to give your custom controls the ability to persist application settings when hosted in third-party applications.

How to: Create Application Settings
Demonstrates creating new application settings that are persisted between application sessions.

How to: Validate Application Settings
Demonstrates validating application settings before they are persisted.

Related topics

Windows Forms Configuration Section Documents the settings to enable High DPI support in Windows Forms Application starting with the .NET Framework 4.7.

See also

- Windows Forms

Application Settings Overview

3/9/2021 • 4 minutes to read • Edit Online

This article discusses how to create and store settings data on behalf of your application and your users.

The Application Settings feature of Windows Forms makes it easy to create, store, and maintain custom application and user preferences on the client computer. With Windows Forms application settings, you can store not only application data such as database connection strings, but also user-specific data, such as user application preferences. Using Visual Studio or custom managed code, you can create new settings, read them from and write them to disk, bind them to properties on your forms, and validate settings data prior to loading and saving.

Application settings enables developers to save state in their application using very little custom code, and is a replacement for dynamic properties in previous versions of the .NET Framework. Application settings contains many improvements over dynamic properties, which are read-only, late-bound, and require more custom programming. The dynamic property classes have been retained in .NET Framework 2.0, but they are just shell classes that thinly wrap the application settings classes.

What Are Application Settings

Your Windows Forms applications will often require data that's critical to running the application, but which you don't want to include directly in the application's code. If your application uses a Web Service or a database server, you may want to store this information in a separate file, so that you can change it in the future without recompiling. Similarly, your applications may require storing data that is specific to the current user. Most applications, for example, have user preferences that customize the application's appearance and behavior.

Application settings addresses both needs by providing an easy way to store both application-scoped and user-scoped settings on the client computer. Using Visual Studio or a code editor, you define a setting for a given property by specifying its name, data type, and scope (application or user). You can even place related settings into named groups for easier use and readability. Once defined, these settings are persisted and read back into memory automatically at run time. A pluggable architecture enables the persistence mechanism to be changed, but by default, the local file system is used.

Application settings works by persisting data as XML to different configuration (.config) files, corresponding to whether the setting is application-scoped or user-scoped. In most cases, the application-scoped settings are read-only; because they are program information, you will typically not need to overwrite them. By contrast, user-scoped settings can be read and written safely at run time, even if your application runs under partial trust. For more information about partial trust, see Security in Windows Forms Overview.

Settings are stored as XML fragments in configuration files. Application-scoped settings are represented by the `<applicationSettings>` element, and generally are placed in *app*.exe.config, where *app* is the name of your main executable file. User-scoped settings are represented by the `<userSettings>` element and are placed in *user*.config, where *user* is the user name of the person currently running the application. You must deploy the *app*.exe.config file with your application; the settings architecture will create the *user*.config files on demand the first time the application saves settings for that user. You can also define a `<userSettings>` block within *app*.exe.config to provide default values for user-scoped settings.

Custom controls can also save their own settings by implementing the IPersistComponentSettings interface, which exposes the SaveSettings method. The Windows Forms ToolStrip control implements this interface to save the position of toolbars and toolbar items between application sessions. For more information about custom controls and application settings, see Application Settings for Custom Controls.

Limitations of Application Settings

You cannot use application settings in an unmanaged application that hosts the .NET Framework. Settings will not work in such environments as Visual Studio add-ins, C++ for Microsoft Office, control hosting in Internet Explorer, or Microsoft Outlook add-ins and projects.

You currently cannot bind to some properties in Windows Forms. The most notable example is the ClientSize property, as binding to this property would cause unpredictable behavior at run time. You can usually work around these issues by saving and loading these settings programmatically.

Application settings has no built-in facility for encrypting information automatically. You should never store security-related information, such as database passwords, in clear text. If you want to store such sensitive information, you as the application developer are responsible for making sure it is secure. If you want to store connection strings, we recommend that you use Windows Integrated Security and not resort to hard-coding passwords into the URL. For more information, see Code Access Security and ADO.NET.

Getting Started with Application Settings

If you use Visual Studio, you can define settings within the Windows Forms Designer using the **(ApplicationSettings)** property in the **Properties** window. When you define settings this way, Visual Studio automatically creates a custom managed wrapper class that associates each setting with a class property. Visual Studio also takes care of binding the setting to a property on a form or control so that the control's settings are restored automatically when its form is displayed, and saved automatically when the form is closed.

If you want more detailed control over your settings, you can define your own custom applications settings wrapper class. This is accomplished by deriving a class from ApplicationSettingsBase, adding a property that corresponds to each setting, and applying special attributes to these properties. For details about creating wrapper classes, see Application Settings Architecture.

You can also use the Binding class to bind settings programmatically to properties on forms and controls.

See also

- ApplicationSettingsBase
- SettingsProvider
- LocalFileSettingsProvider
- IPersistComponentSettings
- How to: Validate Application Settings
- Managing Application Settings (.NET)
- How To: Read Settings at Run Time With C#
- Using Application Settings and User Settings
- Application Settings Architecture
- Application Settings for Custom Controls

Application Settings Architecture

3/9/2021 • 9 minutes to read • Edit Online

This topic describes how the Application Settings architecture works, and explores advanced features of the architecture, such as grouped settings and settings keys.

The application settings architecture supports defining strongly typed settings with either application or user scope, and persisting the settings between application sessions. The architecture provides a default persistence engine for saving settings to and loading them from the local file system. The architecture also defines interfaces for supplying a custom persistence engine.

Interfaces are provided that enable custom components to persist their own settings when they are hosted in an application. By using settings keys, components can keep settings for multiple instances of the component separate.

Defining Settings

The application settings architecture is used within both ASP.NET and Windows Forms, and it contains a number of base classes that are shared across both environments. The most important is SettingsBase, which provides access to settings through a collection, and provides low-level methods for loading and saving settings. Each environment implements its own class derived from SettingsBase to provide additional settings functionality for that environment. In a Windows Forms-based application, all application settings must be defined on a class derived from the ApplicationSettingsBase class, which adds the following functionality to the base class:

- Higher-level loading and saving operations
- Support for user-scoped settings
- Reverting a user's settings to the predefined defaults
- Upgrading settings from a previous application version
- Validating settings, either before they are changed or before they are saved

The settings can be described using a number of attributes defined within the System.Configuration namespace; these are described in Application Settings Attributes. When you define a setting, you must apply it with either ApplicationScopedSettingAttribute or UserScopedSettingAttribute, which describes whether the setting applies to the entire application or just to the current user.

The following code example defines a custom settings class with a single setting, `BackgroundColor`.

```
using System;
using System.Configuration;
using System.Drawing;

public class MyUserSettings : ApplicationSettingsBase
{
    [UserScopedSetting()]
    [DefaultSettingValue("white")]
    public Color BackgroundColor
    {
        get
        {
            return ((Color)this["BackgroundColor"]);
        }
        set
        {
            this["BackgroundColor"] = (Color)value;
        }
    }
}
```

```
Imports System.Configuration

Public Class MyUserSettings
    Inherits ApplicationSettingsBase
    <UserScopedSetting()> _
    <DefaultSettingValue("white")> _
    Public Property BackgroundColor() As Color
        Get
            BackgroundColor = Me("BackgroundColor")
        End Get

        Set(ByVal value As Color)
            Me("BackgroundColor") = value
        End Set
    End Property
End Class
```

Settings Persistence

The ApplicationSettingsBase class does not itself persist or load settings; this job falls to the settings provider, a class that derives from SettingsProvider. If a derived class of ApplicationSettingsBase does not specify a settings provider through the SettingsProviderAttribute, then the default provider, LocalFileSettingsProvider, is used.

The configuration system that was originally released with the .NET Framework supports providing static application configuration data through either the local computer's machine.config file or within an `app.` exe.config file that you deploy with your application. The LocalFileSettingsProvider class expands this native support in the following ways:

- Application-scoped settings can be stored in either the machine.config or `app.` exe.config files. Machine.config is always read-only, while `app` .exe.config is restricted by security considerations to read-only for most applications.

- User-scoped settings can be stored in `app` .exe.config files, in which case they are treated as static defaults.

- Non-default user-scoped settings are stored in a new file, *user*.config, where *user* is the user name of the person currently executing the application. You can specify a default for a user-scoped setting with DefaultSettingValueAttribute. Because user-scoped settings often change during application execution, `user` .config is always read/write.

All three configuration files store settings in XML format. The top-level XML element for application-scoped settings is `<appSettings>`, while `<userSettings>` is used for user-scoped settings. An `app`.exe.config file which contains both application-scoped settings and defaults for user-scoped settings would look like this:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup, 
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="WindowsApplication1.Properties.Settings" 
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, 
PublicKeyToken=b77a5c561934e089" />
        </sectionGroup>
        <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, 
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="WindowsApplication1.Properties.Settings" 
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, 
PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalUser" />
        </sectionGroup>
    </configSections>
    <applicationSettings>
        <WindowsApplication1.Properties.Settings>
            <setting name="Cursor" serializeAs="String">
                <value>Default</value>
            </setting>
            <setting name="DoubleBuffering" serializeAs="String">
                <value>False</value>
            </setting>
        </WindowsApplication1.Properties.Settings>
    </applicationSettings>
    <userSettings>
        <WindowsApplication1.Properties.Settings>
            <setting name="FormTitle" serializeAs="String">
                <value>Form1</value>
            </setting>
            <setting name="FormSize" serializeAs="String">
                <value>595, 536</value>
            </setting>
        </WindowsApplication1.Properties.Settings>
    </userSettings>
</configuration>
```

For a definition of the elements within the application settings section of a configuration file, see Application Settings Schema.

Settings Bindings

Application settings uses the Windows Forms data binding architecture to provide two-way communication of settings updates between the settings object and components. If you use Visual Studio to create application settings and assign them to component properties, these bindings are generated automatically.

You can only bind an application setting to a component that supports the IBindableComponent interface. Also, the component must implement a change event for a specific bound property, or notify application settings that the property has changed through the INotifyPropertyChanged interface. If the component does not implement IBindableComponent and you are binding through Visual Studio, the bound properties will be set the first time, but will not update. If the component implements IBindableComponent but does not support property change notifications, the binding will not update in the settings file when the property is changed.

Some Windows Forms components, such as ToolStripItem, do not support settings bindings.

Settings Serialization

When LocalFileSettingsProvider must save settings to disk, it performs the following actions:

1. Uses reflection to examine all of the properties defined on your ApplicationSettingsBase derived class, finding those that are applied with either ApplicationScopedSettingAttribute or UserScopedSettingAttribute.

2. Serializes the property to disk. It first attempts to call the ConvertToString or ConvertFromString on the type's associated TypeConverter. If this does not succeed, it uses XML serialization instead.

3. Determines which settings go in which files, based on the setting's attribute.

If you implement your own settings class, you can use the SettingsSerializeAsAttribute to mark a setting for either binary or custom serialization using the SettingsSerializeAs enumeration. For more information on creating your own settings class in code, see How to: Create Application Settings.

Settings File Locations

The location of the `app`.exe.config and *user*.config files will differ based on how the application is installed. For a Windows Forms-based application copied onto the local computer, `app`.exe.config will reside in the same directory as the base directory of the application's main executable file, and *user*.config will reside in the location specified by the Application.LocalUserAppDataPath property. For an application installed by means of ClickOnce, both of these files will reside in the ClickOnce Data Directory underneath %InstallRoot%\Documents and Settings*username*\Local Settings.

The storage location of these files is slightly different if a user has enabled roaming profiles, which enables a user to define different Windows and application settings when they are using other computers within a domain. In that case, both ClickOnce applications and non-ClickOnce applications will have their `app`.exe.config and *user*.config files stored under %InstallRoot%\Documents and Settings*username*\Application Data.

For more information about how the Application Settings feature works with the new deployment technology, see ClickOnce and Application Settings. For more information about the ClickOnce Data Directory, see Accessing Local and Remote Data in ClickOnce Applications.

Application Settings and Security

Application settings are designed to work in partial trust, a restricted environment that is the default for Windows Forms applications hosted over the Internet or an intranet. No special permissions beyond partial trust are needed to use application settings with the default settings provider.

When application settings are used in a ClickOnce application, the `user`.config file is stored in the ClickOnce data directory. The size of the application's `user`.config file cannot exceed the data directory quota set by ClickOnce. For more information, see ClickOnce and Application Settings.

Custom Settings Providers

In the Application Settings architecture, there is a loose coupling between the applications settings wrapper class, derived from ApplicationSettingsBase, and the associated settings provider or providers, derived from SettingsProvider. This association is defined only by the SettingsProviderAttribute applied to the wrapper class or its individual properties. If a settings provider is not explicitly specified, the default provider, LocalFileSettingsProvider, is used. As a result, this architecture supports creating and using custom settings providers.

For example, suppose that you want to develop and use `SqlSettingsProvider`, a provider that will store all settings data in a Microsoft SQL Server database. Your SettingsProvider-derived class would receive this information in its `Initialize` method as a parameter of type System.Collections.Specialized.NameValueCollection. You would then implement the GetPropertyValues method to retrieve your settings from the data store, and SetPropertyValues to save them. Your provider can use the SettingsPropertyCollection supplied to GetPropertyValues to determine the property's name, type, and scope, as well as any other settings attributes defined for that property.

Your provider will need to implement one property and one method whose implementations may not be obvious. The ApplicationName property is an abstract property of SettingsProvider; you should program it to return the following:

```
public override string ApplicationName
{
    get
    {
        return (System.Reflection.Assembly.GetExecutingAssembly().GetName().Name);
    }
    set
    {
        // Do nothing.
    }
}
```

```
Public Overrides Property ApplicationName() As String
    Get
        ApplicationName = System.Reflection.Assembly.GetExecutingAssembly().GetName().Name
    End Get
    Set(ByVal value As String)
        ' Do nothing.
    End Set
End Property
```

Your derived class must also implement an `Initialize` method that takes no arguments and returns no value. This method is not defined by SettingsProvider.

Finally, you implement IApplicationSettingsProvider on your provider to provide support for refreshing settings, reverting settings to their defaults, and upgrading settings from one application version to another.

Once you have implemented and compiled your provider, you need to instruct your settings class to use this provider instead of the default. You accomplish this through the SettingsProviderAttribute. If applied to an entire settings class, the provider is used for each setting that the class defines; if applied to individual settings, Application Settings architecture uses that provider for those settings only, and uses LocalFileSettingsProvider for the rest. The following code example shows how to instruct the settings class to use your custom provider.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Configuration;

namespace ApplicationSettingsArchitectureCS
{
    [SettingsProvider("SqlSettingsProvider")]
    class CustomSettings : ApplicationSettingsBase
    {
        // Implementation goes here.
    }
}
```

```
Imports System.Configuration

<SettingsProvider("SqlSettingsProvider")> _
Public Class CustomSettings
    Inherits ApplicationSettingsBase

    ' Implementation goes here.
End Class
```

A provider may be called from multiple threads simultaneously, but it will always write to the same storage location; therefore, the Application Settings architecture will only ever instantiate a single instance of your provider class.

> **IMPORTANT**
>
> You should ensure that your provider is thread-safe, and only allows one thread at a time to write to the configuration files.

Your provider does not need to support all of the settings attributes defined in the System.Configuration namespace, though it must at a minimum support ApplicationScopedSettingAttribute and UserScopedSettingAttribute, and should also support DefaultSettingValueAttribute. For those attributes that it does not support, your provider should just fail without notification; it should not throw an exception. If the settings class uses an invalid combination of attributes, however — such as applying ApplicationScopedSettingAttribute and UserScopedSettingAttribute to the same setting — your provider should throw an exception and cease operation.

See also

- ApplicationSettingsBase
- SettingsProvider
- LocalFileSettingsProvider
- Application Settings Overview
- Application Settings for Custom Controls
- ClickOnce and Application Settings
- Application Settings Schema

Application Settings Attributes

11/3/2020 • 2 minutes to read • Edit Online

The Application Settings architecture provides many attributes that can be applied either to the applications settings wrapper class or its individual properties. These attributes are examined at run time by the application settings infrastructure, often specifically the settings provider, in order to tailor its functioning to the stated needs of the custom wrapper.

The following table lists the attributes that can be applied to the application settings wrapper class, this class's individual properties, or both. By definition, only a single scope attribute—**UserScopedSettingAttribute** or **ApplicationScopedSettingAttribute**—must be applied to each and every settings property.

NOTE

A custom settings provider, derived from the SettingsProvider class, is only required to recognize the following three attributes: **ApplicationScopedSettingAttribute**, **UserScopedSettingAttribute**, and **DefaultSettingValueAttribute**.

ATTRIBUTE	TARGET	DESCRIPTION
SettingsProviderAttribute	Both	Specifies the short name of the settings provider to use for persistence. If this attribute is not supplied, the default provider, LocalFileSettingsProvider, is assumed.
UserScopedSettingAttribute	Both	Defines a property as a user-scoped application setting.
ApplicationScopedSettingAttribute	Both	Defines a property as an application-scoped application setting.
DefaultSettingValueAttribute	Property	Specifies a string that can be deserialized by the provider into the hard-coded default value for this property. The LocalFileSettingsProvider does not require this attribute, and will override any value provided by this attribute if there is a value already persisted.
SettingsDescriptionAttribute	Property	Provides the descriptive test for an individual setting, used primarily by run-time and design-time tools.
SettingsGroupNameAttribute	Class	Provides an explicit name for a settings group. If this attribute is missing, ApplicationSettingsBase uses the wrapper class name.

ATTRIBUTE	TARGET	DESCRIPTION
SettingsGroupDescriptionAttribute	Class	Provides the descriptive test for a settings group, used primarily by run-time and design-time tools.
SettingsManageabilityAttribute	Both	Specifies zero or more manageability services that should be provided to the settings group or property. The available services are described by the SettingsManageability enumeration.
SpecialSettingAttribute	Property	Indicates that a setting belongs to a special, predefined category, such as a connection string, that suggests special processing by the settings provider. The predefined categories for this attribute are defined by the SpecialSetting enumeration.
SettingsSerializeAsAttribute	Both	Specifies a preferred serialization mechanism for a settings group or property. The available serialization mechanisms are defined by the SettingsSerializeAs enumeration.
NoSettingsVersionUpgradeAttribute	Property	Specifies that a settings provider should disable all application upgrade functionality for the marked property.

Class indicates that the attribute can be applied only to an application settings wrapper class. *Property* indicates that the attribute can be applied only settings properties. *Both* indicates that the attribute can be applied at either level.

See also

- ApplicationSettingsBase
- SettingsProvider
- Application Settings Architecture
- How to: Create Application Settings

Application Settings for Custom Controls

11/3/2020 • 2 minutes to read • Edit Online

You must complete certain tasks to give your custom controls the ability to persist application settings when the controls are hosted in third-party applications.

Most of the documentation about the Application Settings feature is written under the assumption that you are creating a standalone application. However, if you are creating a control that other developers will host in their applications, you need to take a few additional steps for your control to persist its settings properly.

Application Settings and Custom Controls

For your control to properly persist its settings, it must encapsulate the process by creating its own dedicated applications settings wrapper class, derived from ApplicationSettingsBase. Additionally, the main control class must implement the IPersistComponentSettings. The interface contains several properties as well as two methods, LoadComponentSettings and SaveComponentSettings. If you add your control to a form using the **Windows Forms Designer** in Visual Studio, Windows Forms will call LoadComponentSettings automatically when the control is initialized; you must call SaveComponentSettings yourself in the `Dispose` method of your control.

In addition, you should implement the following in order for application settings for custom controls to work properly in design-time environments such as Visual Studio:

1. A custom application settings class with a constructor that takes an IComponent as a single parameter. Use this class to save and load all of your application settings. When you create a new instance of this class, pass your custom control using the constructor.
2. Create this custom settings class after the control has been created and placed on a form, such as in the form's Load event handler.

For instructions on creating a custom settings class, see How to: Create Application Settings.

Settings Keys and Shared Settings

Some controls can be used multiple times within the same form. Most of the time, you will want these controls to persist their own individual settings. With the SettingsKey property on IPersistComponentSettings, you can supply a unique string that acts to disambiguate multiple versions of a control on a form.

The simplest way to implement SettingsKey is to use the Name property of the control for the SettingsKey. When you load or save the control's settings, you pass the value of SettingsKey on to the SettingsKey property of the ApplicationSettingsBase class. Application Settings uses this unique key when it persists the user's settings to XML. The following code example shows how a `<userSettings>` section may look for an instance of a custom control named `CustomControl1` that saves a setting for its `Text` property.

```
<userSettings>
    <CustomControl1>
        <setting name="Text" serializedAs="string">
            <value>Hello, World</value>
        </setting>
    </CustomControl1>
</userSettings>
```

Any instances of a control that do not supply a value for SettingsKey will share the same settings.

See also

- ApplicationSettingsBase
- IPersistComponentSettings
- Application Settings Architecture

Using Application Settings and User Settings

11/3/2020 • 2 minutes to read • Edit Online

Starting with the .NET Framework 2.0, you can create and access values that are persisted between application execution sessions. These values are called *settings*. Settings can represent user preferences, or valuable information the application needs to use. For example, you might create a series of settings that store user preferences for the color scheme of an application. Or you might store the connection string that specifies a database that your application uses. Settings allow you to both persist information that is critical to the application outside the code, and to create profiles that store the preferences of individual users.

The topics in this section describe how to use settings at design time and run time.

In This Section

How To: Create a New Setting at Design Time

Explains how to use Visual Studio to create a new setting for an application.

How To: Change the Value of an Existing Setting at Design Time

Describes how to use Visual Studio to change the value of an existing setting.

How To: Change the Value of a Setting Between Application Sessions

Details how to change the value of a setting in a compiled application between application sessions.

How To: Read Settings at Run Time With C#

Describes how to use code to read settings with C#.

How To: Write User Settings at Run Time with C#

Explains how to use code to write and save the values of user settings with C#.

How To: Add Multiple Sets of Settings To Your Application in C#

Details how to add multiple sets of settings to an application with C#.

See also

- Application Settings for Windows Forms

How To: Create a new setting at design time

11/3/2020 • 2 minutes to read • Edit Online

You can create a new setting at design time by using the Settings designer in Visual Studio. The Settings designer is a grid-style interface that allows you to create new settings and specify properties for those settings. You must specify Name, Value, Type and Scope for your new settings. Once a setting is created, it is accessible in code.

Create a new setting at design time in C#

1. Open Visual Studio.
2. In **Solution Explorer**, expand the **Properties** node of your project.
3. Double-click the .settings file in which you want to add a new setting. The default name for this file is Settings.settings.
4. In the Settings designer, set the **Name**, **Value**, **Type**, and **Scope** for your setting. Each row represents a single setting.

Create a new setting at design time in Visual Basic

1. Open Visual Studio.
2. In **Solution Explorer**, right-click your project node and choose **Properties**.
3. In the **Properties** page, select the **Settings** tab.
4. In the Settings designer, set the **Name**, **Value**, **Type**, and **Scope** for your setting. Each row represents a single setting.

See also

- Using Application Settings and User Settings
- Application Settings Overview
- How To: Change the Value of an Existing Setting at Design Time

How To: Change the Value of an Existing Setting at Design Time

11/3/2020 • 2 minutes to read • Edit Online

You can use Visual Studio to edit the values of existing settings in your project.

To Change the Value of an Existing Setting at Design Time in C#

1. In **Solution Explorer**, expand the **Properties** node of your project.

2. Double-click the .settings file in which you want to add a new setting. The default name for this file is Settings.settings.

3. In the Settings designer, find the setting for which you want to change the value and type the new value in the Value column.

To Change the Value of an Existing Setting at Design Time in Visual Basic

1. In **Solution Explorer**, right-click your project node and choose **Properties**.

2. In the **Properties** page, select the **Settings** tab.

3. In the Settings designer, find the setting for which you want to change the value and type the new value in the Value column.

See also

- Using Application Settings and User Settings
- How To: Create a New Setting at Design Time
- Application Settings Overview

How To: Change the Value of a Setting Between Application Sessions

11/3/2020 • 2 minutes to read • Edit Online

At times, you might want to change the value of a setting between application sessions after the application has been compiled and deployed. For example, you might want to change a connection string to point to the correct database location. Since design-time tools are not available after the application has been compiled and deployed, you must change the setting value manually in the file.

To Change the Value of a Setting Between Application Sessions

1. Using Microsoft Notepad or some other text or XML editor, open the .config file associated with your application.

2. Locate the entry for the setting you want to change. It should look similar to the example presented below.

```
<setting name="Setting1" serializeAs="String" >
   <value>My Setting Value</value>
</setting>
```

3. Type a new value for your setting and save the file.

See also

- Using Application Settings and User Settings
- Application Settings Overview

How To: Read Settings at Run Time With C#

11/3/2020 • 2 minutes to read • Edit Online

You can read both Application-scoped and User-scoped settings at run time via the Properties object. The Properties object exposes all of the default settings for the project via the Properties.Settings.Default member in the default namespace of the project they are defined in.

To Read Settings at Run Time with C#

Access the appropriate setting via the Properties.Settings.Default member. The following example shows how to assign a setting named `myColor` to a BackColor property. It requires you to have previously created a Settings file containing a setting named `myColor` of type `System.Drawing.Color`. For information about creating a Settings file, see How To: Create a New Setting at Design Time.

```
this.BackColor = Properties.Settings.Default.myColor;
```

See also

- Using Application Settings and User Settings
- How To: Write User Settings at Run Time with C#
- Application Settings Overview

How To: Write User Settings at Run Time with C#

11/3/2020 • 2 minutes to read • Edit Online

Settings that are application-scoped are read-only, and can only be changed at design time or by altering the .config file in between application sessions. Settings that are user-scoped, however, can be written at run time just as you would change any property value. The new value persists for the duration of the application session. You can persist the changes to the settings between application sessions by calling the Save method.

How To: Write and Persist User Settings at Run Time with C#

1. Access the setting and assign it a new value as shown in this example:

   ```
   Properties.Settings.Default.myColor = Color.AliceBlue;
   ```

2. If you want to persist the changes to the settings between application sessions, call the Save method as shown in this example:

   ```
   Properties.Settings.Default.Save();
   ```

User settings are saved in a file within a subfolder of the user's local hidden application data folder.

See also

- Using Application Settings and User Settings
- How To: Read Settings at Run Time With C#
- Application Settings Overview

How To: Add Multiple Sets of Settings To Your Application in C#

11/3/2020 • 2 minutes to read • Edit Online

In some cases, you might want to have multiple sets of settings in an application. For example, if you are developing an application where a particular group of settings is expected to change frequently, it might be wise to separate them all into a single file so that the file can be replaced wholesale, leaving other settings unaffected. Visual Studio allows you to add multiple sets of settings to your project. Additional sets of settings can be accessed via the `Properties.Settings` object.

Add an Additional Set of Settings

1. In Visual Studio, from the **Project** menu, choose **Add New Item**.

   The **Add New Item** dialog box opens.

2. In the **Add New Item** dialog box, select **Settings File**, enter a name for the file, and click **Add** to add a new settings file to your solution.

3. In **Solution Explorer**, drag the new Settings file into the **Properties** folder. This allows your new settings to be available in code.

4. Add and use settings in this file as you would any other settings file. You can access this group of settings via the `Properties.Settings` object.

See also

- Using Application Settings and User Settings
- Application Settings Overview

How to: Create Application Settings

3/9/2021 • 2 minutes to read • Edit Online

Using managed code, you can create new application settings and bind them to properties on your form or your form's controls, so that these settings are loaded and saved automatically at run time.

In the following procedure, you manually create a wrapper class that derives from ApplicationSettingsBase. To this class you add a publicly accessible property for each application setting that you want to expose.

You can also perform this procedure using minimal code in the Visual Studio designer. Also see How to: Create Application Settings Using the Designer.

To create new Application Settings programmatically

1. Add a new class to your project, and rename it. For this procedure, we will call this class `MyUserSettings`. Change the class definition so that the class derives from ApplicationSettingsBase.

2. Define a property on this wrapper class for each application setting you require, and apply that property with either the ApplicationScopedSettingAttribute or UserScopedSettingAttribute, depending on the scope of the setting. For more information about settings scope, see Application Settings Overview. By now, your code should look like this:

```
using System;
using System.Configuration;
using System.Drawing;

public class MyUserSettings : ApplicationSettingsBase
{
    [UserScopedSetting()]
    [DefaultSettingValue("white")]
    public Color BackgroundColor
    {
        get
        {
            return ((Color)this["BackgroundColor"]);
        }
        set
        {
            this["BackgroundColor"] = (Color)value;
        }
    }
}
```

```
Imports System.Configuration

Public Class MyUserSettings
    Inherits ApplicationSettingsBase
    <UserScopedSetting()> _
    <DefaultSettingValue("white")> _
    Public Property BackgroundColor() As Color
        Get
            BackgroundColor = Me("BackgroundColor")
        End Get

        Set(ByVal value As Color)
            Me("BackgroundColor") = value
        End Set
    End Property
End Class
```

3. Create an instance of this wrapper class in your application. It will commonly be a private member of the main form. Now that you have defined your class, you need to bind it to a property; in this case, the BackColor property of your form. You can accomplish this in your form's `Load` event handler.

```
MyUserSettings mus;

private void Form1_Load(object sender, EventArgs e)
{
    mus = new MyUserSettings();
    mus.BackgroundColor = Color.AliceBlue;
    this.DataBindings.Add(new Binding("BackColor", mus, "BackgroundColor"));
}
```

```
Dim Mus As MyUserSettings

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Mus = New MyUserSettings()
    Mus.BackgroundColor = Color.AliceBlue
    Me.DataBindings.Add(New Binding("BackColor", Mus, "BackgroundColor"))
End Sub
```

4. If you provide a way to change settings at run time, you will need to save the user's current settings to disk when your form closes, or else these changes will be lost.

```
//Make sure to hook up this event handler in the constructor!
//this.FormClosing += new FormClosingEventHandler(Form1_FormClosing);
    void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        mus.Save();
    }
```

```
Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
    Mus.Save()
End Sub
```

You have now successfully created a new application setting and bound it to the specified property.

.NET Framework Security

The default settings provider, LocalFileSettingsProvider, persists information to configuration files as plain text. This limits security to the file access security provided by the operating system for the current user. Because of this, care must be taken with the information stored in configuration files. For example, one common use for application settings is to store connection strings that point to the application's data store. However, because of security concerns, such strings should not include passwords. For more information about connection strings, see SpecialSetting.

See also

- SpecialSettingAttribute
- LocalFileSettingsProvider
- Application Settings Overview
- How to: Validate Application Settings

How to: Validate Application Settings

3/9/2021 • 4 minutes to read • Edit Online

This topic demonstrates how to validate application settings before they are persisted.

Because application settings are strongly typed, you have some confidence that users cannot assign data of an incorrect type to a given setting. However, a user still may attempt to assign a value to a setting that falls outside of acceptable bounds—for example, supplying a birth date that occurs in the future. ApplicationSettingsBase, the parent class of all application settings classes, exposes four events to enable such bounds checking. Handling these events puts all of your validation code in a single location, rather than scattering it throughout your project.

The event you use depends upon when you need to validate your settings, as described in the following table.

EVENT	OCCURRENCE AND USE
SettingsLoaded	Occurs after the initial loading of a settings property group. Use this event to validate initial values for the entire property group before they are used within the application.
SettingChanging	Occurs before the value of a single settings property is changed. Use this event to validate a single property before it is changed. It can provide immediate feedback to users regarding their actions and choices.
PropertyChanged	Occurs after the value of a single settings property is changed. Use this event to validate a single property after it is changed. This event is rarely used for validation unless a lengthy, asynchronous validation process is required.
SettingsSaving	Occurs before the settings property group is stored. Use this event to validate values for the entire property group before they are persisted to disk.

Typically, you will not use all of these events within the same application for validation purposes. For example, it is often possible to fulfill all validation requirements by handling only the SettingChanging event.

An event handler generally performs one of the following actions when it detects an invalid value:

- Automatically supplies a value known to be correct, such as the default value.
- Re-queries the user of server code for information.
- For events raised before their associated actions, such as SettingChanging and SettingsSaving, uses the CancelEventArgs argument to cancel the operation.

For more information about event handling, see Event Handlers Overview.

The following procedures show how to test for a valid birth date using either the SettingChanging or the

SettingsSaving event. The procedures were written under the assumption that you have already created your application settings; in this example, we will perform bounds checking on a setting named `DateOfBirth`. For more information about creating settings, see How to: Create Application Settings.

To obtain the application settings object

- Obtain a reference to the application settings object (the wrapper instance) by completing one of the following bulleted items:

  - If you created your settings using the Visual Studio Application Settings dialog box in the **Property Editor**, you can retrieve the default settings object generated for your language through the following expression.

    ```
    Properties.Settings.Default
    ```

    ```
    MySettings.Default
    ```

    -or-

  - If you are a Visual Basic developer and you created your application settings using the Project Designer, you can retrieve your settings by using the My.Settings Object.

    -or-

  - If you created your settings by deriving from ApplicationSettingsBase directly, you need to instantiate your class manually.

    ```
    MyCustomSettings settings = new MyCustomSettings();
    ```

    ```
    Dim Settings as New MyCustomSettings()
    ```

The following procedures were written under the assumption that the application settings object was obtained by completing the last bulleted item in this procedure.

To validate Application Settings when a setting is changing

1. If you are a C# developer, in your form or control's `Load` event, add an event handler for the SettingChanging event.

   -or-

   If you are a Visual Basic developer, you should declare the `Settings` variable using the `WithEvents` keyword.

   ```
   public void Form1_Load(Object sender, EventArgs e)
   {
       settings.SettingChanging += new SettingChangingEventHandler(MyCustomSettings_SettingChanging);
   }
   ```

   ```
   Public Sub Form1_Load(sender as Object, e as EventArgs)
       AddHandler settings.SettingChanging, AddressOf MyCustomSettings_SettingChanging
   End Sub
   ```

2. Define the event handler, and write the code inside of it to perform bounds checking on the birth date.

```
private void MyCustomSettings_SettingChanging(Object sender, SettingChangingEventArgs e)
{
    if (e.SettingName.Equals("DateOfBirth"))
    {
        var newDate = (DateTime)e.NewValue;
        if (newDate > DateTime.Now)
        {
            e.Cancel = true;
            // Inform the user.
        }
    }
}
```

```
Private Sub MyCustomSettings_SettingChanging(sender as Object, e as SettingChangingEventArgs) Handles
Settings.SettingChanging
    If (e.SettingName.Equals("DateOfBirth")) Then
        Dim NewDate as Date = CType(e.NewValue, Date)
        If (NewDate > Date.Now) Then
            e.Cancel = True
            ' Inform the user.
        End If
    End If
End Sub
```

To validate Application Settings when a Save occurs

1. In your form or control's `Load` event, add an event handler for the SettingsSaving event.

```
public void Form1_Load(Object sender, EventArgs e)
{
    settings.SettingsSaving += new SettingsSavingEventHandler(MyCustomSettings_SettingsSaving);
}
```

```
Public Sub Form1_Load(Sender as Object, e as EventArgs)
    AddHandler settings.SettingsSaving, AddressOf MyCustomSettings_SettingsSaving
End Sub
```

2. Define the event handler, and write the code inside of it to perform bounds checking on the birth date.

```
private void MyCustomSettings_SettingsSaving(Object sender, SettingsSavingEventArgs e)
{
    if (this["DateOfBirth"] > Date.Now) {
        e.Cancel = true;
    }
}
```

```
Private Sub MyCustomSettings_SettingsSaving(Sender as Object, e as SettingsSavingEventArgs)
    If (Me["DateOfBirth"] > Date.Now) Then
        e.Cancel = True
    End If
End Sub
```

See also

- Creating Event Handlers in Windows Forms
- How to: Create Application Settings

Windows Forms Print Support

11/3/2020 • 2 minutes to read • Edit Online

Printing in Windows Forms consists primarily of using the PrintDocument Component component to enable the user to print, and the PrintPreviewDialog Control control, PrintDialog Component and PageSetupDialog Component components to provide a familiar graphical interface to users accustomed to the Windows operating system.

Typically, you create a new instance of the PrintDocument component, set the properties that describe what to print using the PrinterSettings and PageSettings classes, and call the Print method to actually print the document.

During the course of printing from a Windows-based application, the PrintDocument component will show an abort print dialog box to alert users to the fact that printing is occurring and to allow the print job to be canceled.

In This Section

How to: Create Standard Windows Forms Print Jobs
Explains how to use the PrintDocument component to print from a Windows Form.

How to: Capture User Input from a PrintDialog at Run Time
Explains how to modify selected print options programmatically using the PrintDialog component.

How to: Choose the Printers Attached to a User's Computer in Windows Forms
Describes changing the printer to print to using the PrintDialog component at run time.

How to: Print Graphics in Windows Forms
Describes sending graphics to the printer.

How to: Print a Multi-Page Text File in Windows Forms
Describes sending text to the printer.

How to: Complete Windows Forms Print Jobs
Explains how to alert users to the completion of a print job.

How to: Print a Windows Form
Shows how to print a copy of the current form.

How to: Print in Windows Forms Using Print Preview
Shows how to use a PrintPreviewDialog for printing a document.

Related Sections

PrintDocument Component
Explains usage of the PrintDocument component.

PrintDialog Component
Explains usage of the PrintDialog component.

PrintPreviewDialog Control
Explains usage of the PrintPreviewDialog control.

PageSetupDialog Component

Explains usage of the PageSetupDialog component.

System.Drawing.Printing
Describes the classes in the System.Drawing.Printing namespace.

How to: Create Standard Windows Forms Print Jobs

11/3/2020 • 2 minutes to read • Edit Online

The foundation of printing in Windows Forms is the PrintDocument component—more specifically, the PrintPage event. By writing code to handle the PrintPage event, you can specify what to print and how to print it.

To create a print job

1. Add a PrintDocument component to your form.

2. Write code to handle the PrintPage event.

   You will have to code your own printing logic. Additionally, you will have to specify the material to be printed.

   In the following code example, a sample graphic in the shape of a red rectangle is created in the PrintPage event handler to act as material to be printed.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
   e.Graphics.FillRectangle(Brushes.Red, New Rectangle(500, 500, 500, 500))
End Sub
```

```
private void printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
   e.Graphics.FillRectangle(Brushes.Red,
     new Rectangle(500, 500, 500, 500));
}
```

```
private:
   void printDocument1_PrintPage(System::Object ^ sender,
      System::Drawing::Printing::PrintPageEventArgs ^ e)
   {
      e->Graphics->FillRectangle(Brushes::Red,
         Rectangle(500, 500, 500, 500));
   }
```

   (Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.printDocument1.PrintPage += new
   System.Drawing.Printing.PrintPageEventHandler
   (this.printDocument1_PrintPage);
```

```
printDocument1->PrintPage += gcnew
   System::Drawing::Printing::PrintPageEventHandler
   (this, &Form1::printDocument1_PrintPage);
```

   You may also want to write code for the BeginPrint and EndPrint events, perhaps including an integer representing the total number of pages to print that is decremented as each page prints.

NOTE

You can add a PrintDialog component to your form to provide a clean and efficient user interface (UI) to your users. Setting the Document property of the PrintDialog component enables you to set properties related to the print document you are working with on your form. For more information about the PrintDialog component, see PrintDialog Component.

For more information about the specifics of Windows Forms print jobs, including how to create a print job programmatically, see PrintPageEventArgs.

See also

- PrintDocument
- Windows Forms Print Support

How to: Capture User Input from a PrintDialog at Run Time

11/3/2020 • 2 minutes to read • Edit Online

While you can set options related to printing at design time, you will sometimes want to change these options at run time, most likely because of choices made by the user. You can capture user input for printing a document using the PrintDialog and the PrintDocument components.

To change print options programmatically

1. Add a PrintDialog and a PrintDocument component to your form.

2. Set the Document property of the PrintDialog to the PrintDocument added to the form.

```
PrintDialog1.Document = PrintDocument1
```

```
printDialog1.Document = PrintDocument1;
```

```
printDialog1->Document = PrintDocument1;
```

3. Display the PrintDialog component by using the ShowDialog method.

```
PrintDialog1.ShowDialog()
```

```
printDialog1.ShowDialog();
```

```
printDialog1->ShowDialog();
```

4. The user's printing choices from the dialog will be copied to the PrinterSettings property of the PrintDocument component.

See also

- How to: Print a Multi-Page Text File in Windows Forms
- Windows Forms Print Support

How to: Choose the Printers Attached to a User's Computer in Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

Often, users want to choose a printer other than the default printer to print to. You can enable users to choose a printer from among those currently installed by using the PrintDialog component. Through the PrintDialog component, the DialogResult of the PrintDialog component is captured and used to select the printer.

In the following procedure, a text file is selected to be printed to the default printer. The PrintDialog class is then instantiated.

To choose a printer and then print a file

1. Select the printer to be used using the PrintDialog component.

   In the following code example, there are two events being handled. In the first, a Button control's Click event, the PrintDialog class is instantiated and the printer selected by the user is captured in the DialogResult property.

   In the second event, the PrintPage event of the PrintDocument component, a sample document is printed to the printer specified.

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
   Dim PrintDialog1 As New PrintDialog()
   PrintDialog1.Document = PrintDocument1
   Dim result As DialogResult = PrintDialog1.ShowDialog()

   If (result = DialogResult.OK) Then
     PrintDocument1.Print()
   End If

End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
   e.Graphics.FillRectangle(Brushes.Red, New Rectangle(500, 500, 500, 500))
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
   PrintDialog printDialog1 = new PrintDialog();
   printDialog1.Document = printDocument1;
   DialogResult result = printDialog1.ShowDialog();
   if (result == DialogResult.OK)
   {
      printDocument1.Print();
   }
}

private void printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
   e.Graphics.FillRectangle(Brushes.Red,
     new Rectangle(500, 500, 500, 500));
}
```

```
private:
   void button1_Click(System::Object ^ sender,
      System::EventArgs ^ e)
   {
      PrintDialog ^ printDialog1 = gcnew PrintDialog();
      printDialog1->Document = printDocument1;
      System::Windows::Forms::DialogResult result =
         printDialog1->ShowDialog();
      if (result == DialogResult::OK)
      {
         printDocument1->Print();
      }
   }
private:
   void printDocument1_PrintPage(System::Object ^ sender,
      System::Drawing::Printing::PrintPageEventArgs ^ e)
   {
      e->Graphics->FillRectangle(Brushes::Red,
         Rectangle(500, 500, 500, 500));
   }
```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.printDocument1.PrintPage += new
   System.Drawing.Printing.PrintPageEventHandler
   (this.printDocument1_PrintPage);
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->printDocument1->PrintPage += gcnew
   System::Drawing::Printing::PrintPageEventHandler
   (this, &Form1::printDocument1_PrintPage);
this->button1->Click += gcnew
   System::EventHandler(this, &Form1::button1_Click);
```

See also

- Windows Forms Print Support

How to: Print Graphics in Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

Frequently, you will want to print graphics in your Windows-based application. The Graphics class provides methods for drawing objects to a device, such as a screen or printer.

To print graphics

1. Add a PrintDocument component to your form.
2. In the PrintPage event handler, use the Graphics property of the PrintPageEventArgs class to instruct the printer on what kind of graphics to print.

   The following code example shows an event handler used to create a blue ellipse within a bounding rectangle. The rectangle has the following location and dimensions: beginning at 100, 150 with a width of 250 and a height of 250.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
   e.Graphics.FillEllipse(Brushes.Blue, New Rectangle(100, 150, 250, 250))
End Sub
```

```
private void printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
   e.Graphics.FillRectangle(Brushes.Blue,
     new Rectangle(100, 150, 250, 250));
}
```

```
private:
   void printDocument1_PrintPage(System::Object ^ sender,
      System::Drawing::Printing::PrintPageEventArgs ^ e)
   {
      e->Graphics->FillRectangle(Brushes::Blue,
         Rectangle(100, 150, 250, 250));
   }
```

   (Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.printDocument1.PrintPage += new
   System.Drawing.Printing.PrintPageEventHandler
   (this.printDocument1_PrintPage);
```

```
this->printDocument1->PrintPage += gcnew
   System::Drawing::Printing::PrintPageEventHandler
   (this, &Form1::printDocument1_PrintPage);
```

See also

- Graphics

- Brush
- Windows Forms Print Support

How to: Print a Multi-Page Text File in Windows Forms

3/9/2021 • 5 minutes to read • Edit Online

It is very common for Windows-based applications to print text. The Graphics class provides methods for drawing objects (graphics or text) to a device, such as a screen or printer.

NOTE

The DrawText methods of TextRenderer are not supported for printing. You should always use the DrawString methods of Graphics, as shown in the following code example, to draw text for printing purposes.

To print text

1. Add a PrintDocument component and a string to your form.

```
private PrintDocument printDocument1 = new PrintDocument();
private string stringToPrint;
```

```
Private printDocument1 As New PrintDocument()
Private stringToPrint As String
```

2. If printing a document, set the DocumentName property to the document you wish to print, and open and read the documents contents to the string you added previously.

```
string docName = "testPage.txt";
string docPath = @"c:\";
printDocument1.DocumentName = docName;
using (FileStream stream = new FileStream(docPath + docName, FileMode.Open))
using (StreamReader reader = new StreamReader(stream))
{
    stringToPrint = reader.ReadToEnd();
}
```

```
Dim docName As String = "testPage.txt"
Dim docPath As String = "c:\"
printDocument1.DocumentName = docName
Dim stream As New FileStream(docPath + docName, FileMode.Open)
Try
    Dim reader As New StreamReader(stream)
    Try
        stringToPrint = reader.ReadToEnd()
    Finally
        reader.Dispose()
    End Try
Finally
    stream.Dispose()
End Try
```

3. In the PrintPage event handler, use the Graphics property of the PrintPageEventArgs class and the document contents to calculate line length and lines per page. After each page is drawn, check to see if it

is the last page, and set the HasMorePages property of the PrintPageEventArgs accordingly. The PrintPage event is raised until HasMorePages is `false`. Also, make sure the PrintPage event is associated with its event-handling method.

In the following code example, the event handler is used to print the contents of the "testPage.txt" file in the same font as is used on the form.

```
private void printDocument1_PrintPage(object sender, PrintPageEventArgs e)
{
    int charactersOnPage = 0;
    int linesPerPage = 0;

    // Sets the value of charactersOnPage to the number of characters
    // of stringToPrint that will fit within the bounds of the page.
    e.Graphics.MeasureString(stringToPrint, this.Font,
        e.MarginBounds.Size, StringFormat.GenericTypographic,
        out charactersOnPage, out linesPerPage);

    // Draws the string within the bounds of the page
    e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
        e.MarginBounds, StringFormat.GenericTypographic);

    // Remove the portion of the string that has been printed.
    stringToPrint = stringToPrint.Substring(charactersOnPage);

    // Check to see if more pages are to be printed.
    e.HasMorePages = (stringToPrint.Length > 0);
}
```

```
Private Sub printDocument1_PrintPage(ByVal sender As Object, _
    ByVal e As PrintPageEventArgs)

    Dim charactersOnPage As Integer = 0
    Dim linesPerPage As Integer = 0

    ' Sets the value of charactersOnPage to the number of characters
    ' of stringToPrint that will fit within the bounds of the page.
    e.Graphics.MeasureString(stringToPrint, Me.Font, e.MarginBounds.Size, _
        StringFormat.GenericTypographic, charactersOnPage, linesPerPage)

    ' Draws the string within the bounds of the page
    e.Graphics.DrawString(stringToPrint, Me.Font, Brushes.Black, _
        e.MarginBounds, StringFormat.GenericTypographic)

    ' Remove the portion of the string that has been printed.
    stringToPrint = stringToPrint.Substring(charactersOnPage)

    ' Check to see if more pages are to be printed.
    e.HasMorePages = stringToPrint.Length > 0

End Sub
```

4. Call the Print method to raise the PrintPage event.

```
printDocument1.Print();
```

```
printDocument1.Print()
```

Example

```
using System;
using System.Drawing;
using System.IO;
using System.Drawing.Printing;
using System.Windows.Forms;

namespace PrintApp
{
    public class Form1 : Form
    {
        private Button printButton;
        private PrintDocument printDocument1 = new PrintDocument();
        private string stringToPrint;
        public Form1()
        {
            this.printButton = new System.Windows.Forms.Button();
            this.printButton.Location = new System.Drawing.Point(12, 51);
            this.printButton.Size = new System.Drawing.Size(75, 23);
            this.printButton.Text = "Print";
            this.printButton.Click += new System.EventHandler(this.printButton_Click);
            this.ClientSize = new System.Drawing.Size(292, 266);
            this.Controls.Add(this.printButton);

            // Associate the PrintPage event handler with the PrintPage event.
            printDocument1.PrintPage +=
                new PrintPageEventHandler(printDocument1_PrintPage);
        }

        private void ReadFile()
        {
            string docName = "testPage.txt";
            string docPath = @"c:\";
            printDocument1.DocumentName = docName;
            using (FileStream stream = new FileStream(docPath + docName, FileMode.Open))
            using (StreamReader reader = new StreamReader(stream))
            {
                stringToPrint = reader.ReadToEnd();
            }
        }

        private void printDocument1_PrintPage(object sender, PrintPageEventArgs e)
        {
            int charactersOnPage = 0;
            int linesPerPage = 0;

            // Sets the value of charactersOnPage to the number of characters
            // of stringToPrint that will fit within the bounds of the page.
            e.Graphics.MeasureString(stringToPrint, this.Font,
                e.MarginBounds.Size, StringFormat.GenericTypographic,
                out charactersOnPage, out linesPerPage);

            // Draws the string within the bounds of the page
            e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
                e.MarginBounds, StringFormat.GenericTypographic);

            // Remove the portion of the string that has been printed.
            stringToPrint = stringToPrint.Substring(charactersOnPage);

            // Check to see if more pages are to be printed.
            e.HasMorePages = (stringToPrint.Length > 0);
        }

        private void printButton_Click(object sender, EventArgs e)
        {
            ReadFile();
            printDocument1.Print();
        }
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

```
Imports System.Drawing
Imports System.IO
Imports System.Drawing.Printing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private printButton As Button

    Private printDocument1 As New PrintDocument()
    Private stringToPrint As String

    Public Sub New()
        Me.printButton = New System.Windows.Forms.Button()
        Me.printButton.Location = New System.Drawing.Point(12, 51)
        Me.printButton.Size = New System.Drawing.Size(75, 23)
        Me.printButton.Text = "Print"
        Me.ClientSize = New System.Drawing.Size(292, 266)
    End Sub

    Private Sub ReadFile()
        Dim docName As String = "testPage.txt"
        Dim docPath As String = "c:\"
        printDocument1.DocumentName = docName
        Dim stream As New FileStream(docPath + docName, FileMode.Open)
        Try
            Dim reader As New StreamReader(stream)
            Try
                stringToPrint = reader.ReadToEnd()
            Finally
                reader.Dispose()
            End Try
        Finally
            stream.Dispose()
        End Try
    End Sub

    Private Sub printDocument1_PrintPage(ByVal sender As Object, _
        ByVal e As PrintPageEventArgs)

        Dim charactersOnPage As Integer = 0
        Dim linesPerPage As Integer = 0

        ' Sets the value of charactersOnPage to the number of characters
        ' of stringToPrint that will fit within the bounds of the page.
        e.Graphics.MeasureString(stringToPrint, Me.Font, e.MarginBounds.Size, _
            StringFormat.GenericTypographic, charactersOnPage, linesPerPage)

        ' Draws the string within the bounds of the page
        e.Graphics.DrawString(stringToPrint, Me.Font, Brushes.Black, _
            e.MarginBounds, StringFormat.GenericTypographic)

        ' Remove the portion of the string that has been printed.
        stringToPrint = stringToPrint.Substring(charactersOnPage)

        ' Check to see if more pages are to be printed.
        e.HasMorePages = stringToPrint.Length > 0
    End Sub

    Private Sub printButton_Click(ByVal sender As Object, ByVal e As EventArgs)
        ReadFile()
        printDocument1.Print()
    End Sub


    <STAThread()>  _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.SetCompatibleTextRenderingDefault(False)
        Application.Run(New Form1())
    End Sub
End Class
```

Compiling the Code

This example requires:

- A text file named testPage.txt containing the text to print, located in the root of drive C:\. Edit the code to print a different file.

- References to the System, System.Windows.Forms, System.Drawing assemblies.

- For information about building this example from the command line for Visual Basic or Visual C#, see Building from the Command Line or Command-line Building With csc.exe. You can also build this example in Visual Studio by pasting the code into a new project.

See also

- Graphics
- Brush
- Windows Forms Print Support

How to: Complete Windows Forms Print Jobs

11/3/2020 • 2 minutes to read • Edit Online

Frequently, word processors and other applications that involve printing will provide the option to display a message to users that a print job is complete. You can provide this functionality in your Windows Forms by handling the EndPrint event of the PrintDocument component.

The following procedure requires that you have created a Windows-based application with a PrintDocument component on it, which is the standard way of enabling printing from a Windows-based application. For more information about printing from Windows Forms using the PrintDocument component, see How to: Create Standard Windows Forms Print Jobs.

To complete a print job

1. Set the DocumentName property of the PrintDocument component.

```
PrintDocument1.DocumentName = "MyTextFile"
```

```
printDocument1.DocumentName = "MyTextFile";
```

```
printDocument1->DocumentName = "MyTextFile";
```

2. Write code to handle the EndPrint event.

   In the following code example, a message box is displayed, indicating that the document has finished printing.

```
Private Sub PrintDocument1_EndPrint(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintEventArgs) Handles PrintDocument1.EndPrint
   MessageBox.Show(PrintDocument1.DocumentName + " has finished printing.")
End Sub
```

```
private void printDocument1_EndPrint(object sender,
System.Drawing.Printing.PrintEventArgs e)
{
   MessageBox.Show(printDocument1.DocumentName +
      " has finished printing.");
}
```

```
private:
   void printDocument1_EndPrint(System::Object ^ sender,
      System::Drawing::Printing::PrintEventArgs ^ e)
   {
      MessageBox::Show(String::Concat(printDocument1->DocumentName,
         " has finished printing."));
   }
```

   (Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.printDocument1.EndPrint += new
   System.Drawing.Printing.PrintEventHandler
   (this.printDocument1_EndPrint);
```

```
this->printDocument1->EndPrint += gcnew
   System::Drawing::Printing::PrintEventHandler
   (this, &Form1::printDocument1_EndPrint);
```

See also

- PrintDocument
- Windows Forms Print Support

How to: Print a Windows Form

11/3/2020 • 2 minutes to read • Edit Online

As part of the development process, you typically will want to print a copy of your Windows Form. The following code example shows how to print a copy of the current form by using the CopyFromScreen method.

Example

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Printing;

public class Form1 :
    Form
{
    private Button printButton = new Button();
    private PrintDocument printDocument1 = new PrintDocument();

    public Form1()
    {
        printButton.Text = "Print Form";
        printButton.Click += new EventHandler(printButton_Click);
        printDocument1.PrintPage += new PrintPageEventHandler(printDocument1_PrintPage);
        this.Controls.Add(printButton);
    }

    void printButton_Click(object sender, EventArgs e)
    {
        CaptureScreen();
        printDocument1.Print();
    }

    Bitmap memoryImage;

    private void CaptureScreen()
    {
        Graphics myGraphics = this.CreateGraphics();
        Size s = this.Size;
        memoryImage = new Bitmap(s.Width, s.Height, myGraphics);
        Graphics memoryGraphics = Graphics.FromImage(memoryImage);
        memoryGraphics.CopyFromScreen(this.Location.X, this.Location.Y, 0, 0, s);
    }

    private void printDocument1_PrintPage(System.Object sender,
           System.Drawing.Printing.PrintPageEventArgs e)
    {
        e.Graphics.DrawImage(memoryImage, 0, 0);
    }

    public static void Main()
    {
        Application.Run(new Form1());
    }
}
```

```
Imports System.Windows.Forms
Imports System.Drawing
Imports System.Drawing.Printing

Public Class Form1
    Inherits Form
    Private WithEvents printButton As New Button
    Private WithEvents printDocument1 As New PrintDocument

    Public Sub New()
        printButton.Text = "Print Form"
        Me.Controls.Add(printButton)
    End Sub

    Dim memoryImage As Bitmap

    Private Sub CaptureScreen()
        Dim myGraphics As Graphics = Me.CreateGraphics()
        Dim s As Size = Me.Size
        memoryImage = New Bitmap(s.Width, s.Height, myGraphics)
        Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
        memoryGraphics.CopyFromScreen(Me.Location.X, Me.Location.Y, 0, 0, s)
    End Sub

    Private Sub printDocument1_PrintPage(ByVal sender As System.Object, _
       ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
       printDocument1.PrintPage
        e.Graphics.DrawImage(memoryImage, 0, 0)
    End Sub

    Private Sub printButton_Click(ByVal sender As System.Object, ByVal e As _
       System.EventArgs) Handles printButton.Click
        CaptureScreen()
        printDocument1.Print()
    End Sub

    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub
End Class
```

Robust Programming

The following conditions may cause an exception:

- You do not have permission to access the printer.
- There is no printer installed.

.NET Framework Security

In order to run this code example, you must have permission to access the printer you use with your computer.

See also

- PrintDocument
- How to: Render Images with GDI+
- How to: Print Graphics in Windows Forms

How to: Print in Windows Forms Using Print Preview

11/3/2020 • 7 minutes to read • Edit Online

It is very common in Windows Forms programming to offer print preview in addition to printing services. An easy way to add print preview services to your application is to use a PrintPreviewDialog control in combination with the PrintPage event-handling logic for printing a file.

To preview a text document with a PrintPreviewDialog control

1. Add a PrintPreviewDialog, PrintDocument, and two strings to your form.

```
private PrintPreviewDialog printPreviewDialog1 = new PrintPreviewDialog();
private PrintDocument printDocument1 = new PrintDocument();

// Declare a string to hold the entire document contents.
private string documentContents;

// Declare a variable to hold the portion of the document that
// is not printed.
private string stringToPrint;
```

```
Private printPreviewDialog1 As New PrintPreviewDialog()
Private WithEvents printDocument1 As New PrintDocument()

' Declare a string to hold the entire document contents.
Private documentContents As String

' Declare a variable to hold the portion of the document that
' is not printed.
Private stringToPrint As String
```

2. Set the DocumentName property to the document you wish to print, and open and read the document's contents to the string you added previously.

```
private void ReadDocument()
{
    string docName = "testPage.txt";
    string docPath = @"c:\";
    printDocument1.DocumentName = docName;
    using (FileStream stream = new FileStream(docPath + docName, FileMode.Open))
    using (StreamReader reader = new StreamReader(stream))
    {
        documentContents = reader.ReadToEnd();
    }
    stringToPrint = documentContents;
}
```

```
Private Sub ReadDocument()
    Dim docName As String = "testPage.txt"
    Dim docPath As String = "c:\"
    printDocument1.DocumentName = docName
    Dim stream As New FileStream(docPath + docName, FileMode.Open)
    Try
        Dim reader As New StreamReader(stream)
        Try
            documentContents = reader.ReadToEnd()
        Finally
            reader.Dispose()
        End Try
    Finally
        stream.Dispose()
    End Try
    stringToPrint = documentContents

End Sub
```

3. As you would for printing the document, in the PrintPage event handler, use the Graphics property of the PrintPageEventArgs class and the file contents to calculate lines per page and render the document's contents. After each page is drawn, check to see if it is the last page, and set the HasMorePages property of the PrintPageEventArgs accordingly. The PrintPage event is raised until HasMorePages is `false`. When the document has finished rendering, reset the string to be rendered. Also, make sure the PrintPage event is associated with its event-handling method.

NOTE

You may have already completed steps 2 and 3 if you have implemented printing in your application.

In the following code example, the event handler is used to print the "testPage.txt" file in the same font used on the form.

```
void printDocument1_PrintPage(object sender, PrintPageEventArgs e)
{
    int charactersOnPage = 0;
    int linesPerPage = 0;

    // Sets the value of charactersOnPage to the number of characters
    // of stringToPrint that will fit within the bounds of the page.
    e.Graphics.MeasureString(stringToPrint, this.Font,
        e.MarginBounds.Size, StringFormat.GenericTypographic,
        out charactersOnPage, out linesPerPage);

    // Draws the string within the bounds of the page.
    e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
    e.MarginBounds, StringFormat.GenericTypographic);

    // Remove the portion of the string that has been printed.
    stringToPrint = stringToPrint.Substring(charactersOnPage);

    // Check to see if more pages are to be printed.
    e.HasMorePages = (stringToPrint.Length > 0);

    // If there are no more pages, reset the string to be printed.
    if (!e.HasMorePages)
        stringToPrint = documentContents;
}
```

```
Sub printDocument1_PrintPage(ByVal sender As Object, _
    ByVal e As PrintPageEventArgs) Handles printDocument1.PrintPage

    Dim charactersOnPage As Integer = 0
    Dim linesPerPage As Integer = 0

    ' Sets the value of charactersOnPage to the number of characters
    ' of stringToPrint that will fit within the bounds of the page.
    e.Graphics.MeasureString(stringToPrint, Me.Font, e.MarginBounds.Size, _
        StringFormat.GenericTypographic, charactersOnPage, linesPerPage)

    ' Draws the string within the bounds of the page.
    e.Graphics.DrawString(stringToPrint, Me.Font, Brushes.Black, _
        e.MarginBounds, StringFormat.GenericTypographic)

    ' Remove the portion of the string that has been printed.
    stringToPrint = stringToPrint.Substring(charactersOnPage)

    ' Check to see if more pages are to be printed.
    e.HasMorePages = stringToPrint.Length > 0

    ' If there are no more pages, reset the string to be printed.
    If Not e.HasMorePages Then
        stringToPrint = documentContents
    End If

End Sub
```

4. Set the Document property of the PrintPreviewDialog control to the PrintDocument component on the form.

```
printPreviewDialog1.Document = printDocument1;
```

```
printPreviewDialog1.Document = printDocument1
```

5. Call the ShowDialog method on the PrintPreviewDialog control. You would typically call ShowDialog from the Click event-handling method of a button. Calling ShowDialog raises the PrintPage event and renders the output to the PrintPreviewDialog control. When the user clicks the print icon on the dialog, the PrintPage event is raised again, sending the output to the printer instead of the preview dialog. This is why the string is reset at the end of the rendering process in step 3.

   The following code example shows the Click event-handling method for a button on the form. This event-handling method calls the methods to read the document and show the print preview dialog.

```
private void printPreviewButton_Click(object sender, EventArgs e)
{
    ReadDocument();
    printPreviewDialog1.Document = printDocument1;
printPreviewDialog1.ShowDialog();
}
```

```
Private Sub printPreviewButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles printPreviewButton.Click

    ReadDocument()
    printPreviewDialog1.Document = printDocument1
    printPreviewDialog1.ShowDialog()
End Sub
```

Example

```
using System;
using System.Drawing;
using System.IO;
using System.Drawing.Printing;
using System.Windows.Forms;

namespace PrintPreviewApp
{
    public partial class Form1 : Form
    {
        private Button printPreviewButton;

        private PrintPreviewDialog printPreviewDialog1 = new PrintPreviewDialog();
        private PrintDocument printDocument1 = new PrintDocument();

        // Declare a string to hold the entire document contents.
        private string documentContents;

        // Declare a variable to hold the portion of the document that
        // is not printed.
        private string stringToPrint;

        public Form1()
        {
            this.printPreviewButton = new System.Windows.Forms.Button();
            this.printPreviewButton.Location = new System.Drawing.Point(12, 12);
            this.printPreviewButton.Size = new System.Drawing.Size(125, 23);
            this.printPreviewButton.Text = "Print Preview";
            this.printPreviewButton.Click += new System.EventHandler(this.printPreviewButton_Click);
            this.ClientSize = new System.Drawing.Size(292, 266);
            this.Controls.Add(this.printPreviewButton);
            printDocument1.PrintPage +=
                new PrintPageEventHandler(printDocument1_PrintPage);
        }
        private void ReadDocument()
        {
            string docName = "testPage.txt";
            string docPath = @"c:\";
            printDocument1.DocumentName = docName;
            using (FileStream stream = new FileStream(docPath + docName, FileMode.Open))
            using (StreamReader reader = new StreamReader(stream))
            {
                documentContents = reader.ReadToEnd();
            }
            stringToPrint = documentContents;
        }

        void printDocument1_PrintPage(object sender, PrintPageEventArgs e)
        {
            int charactersOnPage = 0;
            int linesPerPage = 0;

            // Sets the value of charactersOnPage to the number of characters
            // of stringToPrint that will fit within the bounds of the page.
            e.Graphics.MeasureString(stringToPrint, this.Font,
                e.MarginBounds.Size, StringFormat.GenericTypographic,
                out charactersOnPage, out linesPerPage);

            // Draws the string within the bounds of the page.
            e.Graphics.DrawString(stringToPrint, this.Font, Brushes.Black,
            e.MarginBounds, StringFormat.GenericTypographic);

            // Remove the portion of the string that has been printed.
            stringToPrint = stringToPrint.Substring(charactersOnPage);

            // Check to see if more pages are to be printed.
            e.HasMorePages = (stringToPrint.Length > 0);

            // If there are no more pages, reset the string to be printed.
            if (!e.HasMorePages)
                stringToPrint = documentContents;
        }
        private void printPreviewButton_Click(object sender, EventArgs e)
        {
            ReadDocument();
            printPreviewDialog1.Document = printDocument1;
        printPreviewDialog1.ShowDialog();
        }

        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

```
Imports System.Drawing
Imports System.IO
Imports System.Drawing.Printing
Imports System.Windows.Forms



Class Form1
    Inherits Form

    Private WithEvents printPreviewButton As Button

    Private printPreviewDialog1 As New PrintPreviewDialog()
    Private WithEvents printDocument1 As New PrintDocument()

    ' Declare a string to hold the entire document contents.
    Private documentContents As String

    ' Declare a variable to hold the portion of the document that
    ' is not printed.
    Private stringToPrint As String

    Public Sub New()
        Me.printPreviewButton = New System.Windows.Forms.Button()
        Me.printPreviewButton.Location = New System.Drawing.Point(12, 12)
        Me.printPreviewButton.Size = New System.Drawing.Size(125, 23)
        Me.printPreviewButton.Text = "Print Preview"
        Me.ClientSize = New System.Drawing.Size(292, 266)
        Me.Controls.Add(Me.printPreviewButton)

    End Sub

    Private Sub ReadDocument()
```

```
        Dim docName As String = "testPage.txt"
        Dim docPath As String = "c:\"
        printDocument1.DocumentName = docName
        Dim stream As New FileStream(docPath + docName, FileMode.Open)
        Try
            Dim reader As New StreamReader(stream)
            Try
                documentContents = reader.ReadToEnd()
            Finally
                reader.Dispose()
            End Try
        Finally
            stream.Dispose()
        End Try
        stringToPrint = documentContents

    End Sub

    Sub printDocument1_PrintPage(ByVal sender As Object, _
        ByVal e As PrintPageEventArgs) Handles printDocument1.PrintPage

        Dim charactersOnPage As Integer = 0
        Dim linesPerPage As Integer = 0

        ' Sets the value of charactersOnPage to the number of characters
        ' of stringToPrint that will fit within the bounds of the page.
        e.Graphics.MeasureString(stringToPrint, Me.Font, e.MarginBounds.Size, _
            StringFormat.GenericTypographic, charactersOnPage, linesPerPage)

        ' Draws the string within the bounds of the page.
        e.Graphics.DrawString(stringToPrint, Me.Font, Brushes.Black, _
            e.MarginBounds, StringFormat.GenericTypographic)

        ' Remove the portion of the string that has been printed.
        stringToPrint = stringToPrint.Substring(charactersOnPage)

        ' Check to see if more pages are to be printed.
        e.HasMorePages = stringToPrint.Length > 0

        ' If there are no more pages, reset the string to be printed.
        If Not e.HasMorePages Then
            stringToPrint = documentContents
        End If

    End Sub

    Private Sub printPreviewButton_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles printPreviewButton.Click

        ReadDocument()
        printPreviewDialog1.Document = printDocument1
        printPreviewDialog1.ShowDialog()
    End Sub

    <STAThread()>  _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.SetCompatibleTextRenderingDefault(False)
        Application.Run(New Form1())

    End Sub
End Class
```

Compiling the Code

This example requires:

- References to the System, System.Windows.Forms, System.Drawing assemblies.

See also

- How to: Print a Multi-Page Text File in Windows Forms
- Windows Forms Print Support
- More Secure Printing in Windows Forms

Drag-and-Drop Operations and Clipboard Support

11/3/2020 • 2 minutes to read • Edit Online

You can enable user drag-and-drop operations within a Windows-based application by handling a series of events, most notably the DragEnter, DragLeave, and DragDrop events.

You can also implement user cut/copy/paste support and user data transfer to the Clipboard within your Windows-based applications by using simple method calls.

In This Section

Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms
Explains how to start a drag-and-drop operation.

How to: Perform Drag-and-Drop Operations Between Applications
Illustrates how to accomplish drag-and-drop operations across applications.

How to: Add Data to the Clipboard
Describes a way to programmatically insert information on the Clipboard.

How to: Retrieve Data from the Clipboard
Describes how to access the data stored on the Clipboard.

Related Sections

Drag-and-Drop Functionality in Windows Forms
Describes the methods, events, and classes used to implement drag-and-drop behavior.

QueryContinueDrag
Describes the intricacies of the event that asks permission to continue the drag operation.

DoDragDrop
Describes the intricacies of the method that is central to beginning a drag operation.

Clipboard
Also see How to: Send Data to the Active MDI Child.

Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms

11/3/2020 • 3 minutes to read • Edit Online

To perform drag-and-drop operations within Windows-based applications you must handle a series of events, most notably the DragEnter, DragLeave, and DragDrop events. By working with the information available in the event arguments of these events, you can easily facilitate drag-and-drop operations.

Dragging Data

All drag-and-drop operations begin with dragging. The functionality to enable data to be collected when dragging begins is implemented in the DoDragDrop method.

In the following example, the MouseDown event is used to start the drag operation because it is the most intuitive (most drag-and-drop actions begin with the mouse button being depressed). However, remember that any event could be used to initiate a drag-and-drop procedure.

> **NOTE**
> Certain controls have custom drag-specific events. The ListView and TreeView controls, for example, have an ItemDrag event.

To start a drag operation

1. In the MouseDown event for the control where the drag will begin, use the `DoDragDrop` method to set the data to be dragged and the allowed effect dragging will have. For more information, see Data and AllowedEffect.

   The following example shows how to initiate a drag operation. The control where the drag begins is a Button control, the data being dragged is the string representing the Text property of the Button control, and the allowed effects are either copying or moving.

```
Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs)
Handles Button1.MouseDown
   Button1.DoDragDrop(Button1.Text, DragDropEffects.Copy Or DragDropEffects.Move)
End Sub
```

```
private void button1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{
   button1.DoDragDrop(button1.Text, DragDropEffects.Copy |
      DragDropEffects.Move);
}
```

> **NOTE**
> Any data can be used as a parameter in the `DoDragDrop` method; in the example above, the Text property of the Button control was used (rather than hard-coding a value or retrieving data from a dataset) because the property was related to the location being dragged from (the Button control). Keep this in mind as you incorporate drag-and-drop operations into your Windows-based applications.

While a drag operation is in effect, you can handle the QueryContinueDrag event, which "asks permission" of the system to continue the drag operation. When handling this method, it is also the appropriate point for you to call methods that will have an effect on the drag operation, such as expanding a TreeNode in a TreeView control when the cursor hovers over it.

Dropping Data

Once you have begun dragging data from a location on a Windows Form or control, you will naturally want to drop it somewhere. The cursor will change when it crosses an area of a form or control that is correctly configured for dropping data. Any area within a Windows Form or control can be made to accept dropped data by setting the AllowDrop property and handling the DragEnter and DragDrop events.

To perform a drop

1. Set the AllowDrop property to true.

2. In the `DragEnter` event for the control where the drop will occur, ensure that the data being dragged is of an acceptable type (in this case, Text). The code then sets the effect that will happen when the drop occurs to a value in the DragDropEffects enumeration. For more information, see Effect.

```
Private Sub TextBox1_DragEnter(ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs)
Handles TextBox1.DragEnter
   If (e.Data.GetDataPresent(DataFormats.Text)) Then
     e.Effect = DragDropEffects.Copy
   Else
     e.Effect = DragDropEffects.None
   End If
End Sub
```

```
private void textBox1_DragEnter(object sender,
System.Windows.Forms.DragEventArgs e)
{
   if (e.Data.GetDataPresent(DataFormats.Text))
      e.Effect = DragDropEffects.Copy;
   else
      e.Effect = DragDropEffects.None;
}
```

NOTE

You can define your own DataFormats by specifying your own object as the Object parameter of the SetData method. Be sure, when doing this, that the object specified is serializable. For more information, see ISerializable.

3. In the DragDrop event for the control where the drop will occur, use the GetData method to retrieve the data being dragged. For more information, see Data.

   In the example below, a TextBox control is the control being dragged to (where the drop will occur). The code sets the Text property of the TextBox control equal to the data being dragged.

```
Private Sub TextBox1_DragDrop(ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs)
Handles TextBox1.DragDrop
   TextBox1.Text = e.Data.GetData(DataFormats.Text).ToString
End Sub
```

```
private void textBox1_DragDrop(object sender,
System.Windows.Forms.DragEventArgs e)
{
   textBox1.Text = e.Data.GetData(DataFormats.Text).ToString();
}
```

NOTE

Additionally, you can work with the KeyState property, so that, depending on keys depressed during the drag-and-drop operation, certain effects occur (for example, it is standard to copy the dragged data when the CTRL key is pressed).

See also

- How to: Add Data to the Clipboard
- How to: Retrieve Data from the Clipboard
- Drag-and-Drop Operations and Clipboard Support

How to: Perform Drag-and-Drop Operations Between Applications

11/3/2020 • 2 minutes to read • Edit Online

Performing drag-and-drop operations between applications is no different than enabling this action within an application, as long as both applications involved behave according to the "contract" established between the AllowedEffect and Effect properties.

In the following procedure, you will use a Windows-based application you create and the WordPad word processor that is included with the Windows operating system to perform drag-and-drop operations between applications. WordPad has a certain set of allowed effects for text being dragged and dropped; the Windows-based application you will write code for will work with these effects so that drag-and-drop operations may be completed successfully.

To perform a drag-and-drop procedure between applications

1. Create a new Windows Forms application.
2. Add a TextBox control to your form.
3. Configure the TextBox control to receive dropped data.

   For more information, see Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms.

4. Run your Windows-based application, and while the application is running, run WordPad.

   WordPad is a text editor installed by Windows that allows drag-and-drop operations. It is accessible by pressing the **Start** button, selecting **Run**, and then typing `WordPad` into the text box of the **Run** dialog box and clicking **OK**.

5. Once WordPad is open, type a string of text into it.
6. Using the mouse, select the text, and then drag the selected text over to the TextBox control in your Windows-based application.

   Observe that when you mouse over the TextBox control (and, consequently, raise the DragEnter event), the cursor changes, and you can drop the selected text into the TextBox control.

   Additionally, you can configure your TextBox control to allow text strings to be dragged and dropped into WordPad. For more information, see Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms.

See also

- How to: Add Data to the Clipboard
- How to: Retrieve Data from the Clipboard
- Drag-and-Drop Operations and Clipboard Support

How to: Add Data to the Clipboard

11/3/2020 • 5 minutes to read • Edit Online

The Clipboard class provides methods that you can use to interact with the Windows operating system Clipboard feature. Many applications use the Clipboard as a temporary repository for data. For example, word processors use the Clipboard during cut-and-paste operations. The Clipboard is also useful for transferring data from one application to another.

When you add data to the Clipboard, you can indicate the data format so that other applications can recognize the data if they can use that format. You can also add data to the Clipboard in multiple different formats to increase the number of other applications that can potentially use the data.

A Clipboard format is a string that identifies the format so that an application that uses that format can retrieve the associated data. The DataFormats class provides predefined format names for your use. You can also use your own format names or use the type of an object as its format.

To add data to the Clipboard in one or multiple formats, use the SetDataObject method. You can pass any object to this method, but to add data in multiple formats, you must first add the data to a separate object designed to work with multiple formats. Typically, you will add your data to a DataObject, but you can use any type that implements the IDataObject interface.

In .NET Framework 2.0, you can add data directly to the Clipboard by using new methods designed to make basic Clipboard tasks easier. Use these methods when you work with data in a single, common format such as text.

NOTE

All Windows-based applications share the Clipboard. Therefore, the contents are subject to change when you switch to another application.

The Clipboard class can only be used in threads set to single thread apartment (STA) mode. To use this class, ensure that your `Main` method is marked with the STAThreadAttribute attribute.

An object must be serializable for it to be put on the Clipboard. To make a type serializable, mark it with the SerializableAttribute attribute. If you pass a non-serializable object to a Clipboard method, the method will fail without throwing an exception. For more information about serialization, see System.Runtime.Serialization.

To add data to the Clipboard in a single, common format

1. Use the SetAudio, SetFileDropList, SetImage, or SetText method. These methods are available only in .NET Framework 2.0.

```
// Demonstrates SetAudio, ContainsAudio, and GetAudioStream.
public System.IO.Stream SwapClipboardAudio(
    System.IO.Stream replacementAudioStream)
{
    System.IO.Stream returnAudioStream = null;
    if (Clipboard.ContainsAudio())
    {
        returnAudioStream = Clipboard.GetAudioStream();
        Clipboard.SetAudio(replacementAudioStream);
    }
    return returnAudioStream;
}

// Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList
public System.Collections.Specialized.StringCollection
    SwapClipboardFileDropList(
    System.Collections.Specialized.StringCollection replacementList)
{
    System.Collections.Specialized.StringCollection returnList = null;
    if (Clipboard.ContainsFileDropList())
    {
        returnList = Clipboard.GetFileDropList();
        Clipboard.SetFileDropList(replacementList);
    }
    return returnList;
}

// Demonstrates SetImage, ContainsImage, and GetImage.
public System.Drawing.Image SwapClipboardImage(
    System.Drawing.Image replacementImage)
{
    System.Drawing.Image returnImage = null;
    if (Clipboard.ContainsImage())
    {
        returnImage = Clipboard.GetImage();
        Clipboard.SetImage(replacementImage);
    }
    return returnImage;
}

// Demonstrates SetText, ContainsText, and GetText.
public String SwapClipboardHtmlText(String replacementHtmlText)
{
    String returnHtmlText = null;
    if (Clipboard.ContainsText(TextDataFormat.Html))
    {
        returnHtmlText = Clipboard.GetText(TextDataFormat.Html);
        Clipboard.SetText(replacementHtmlText, TextDataFormat.Html);
    }
    return returnHtmlText;
}
```

```vb
' Demonstrates SetAudio, ContainsAudio, and GetAudioStream.
Public Function SwapClipboardAudio( _
    ByVal replacementAudioStream As System.IO.Stream) _
    As System.IO.Stream

    Dim returnAudioStream As System.IO.Stream = Nothing

    If (Clipboard.ContainsAudio()) Then
        returnAudioStream = Clipboard.GetAudioStream()
        Clipboard.SetAudio(replacementAudioStream)
    End If

    Return returnAudioStream

End Function

' Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList
Public Function SwapClipboardFileDropList(ByVal replacementList _
    As System.Collections.Specialized.StringCollection) _
    As System.Collections.Specialized.StringCollection

    Dim returnList As System.Collections.Specialized.StringCollection _
        = Nothing

    If Clipboard.ContainsFileDropList() Then

        returnList = Clipboard.GetFileDropList()
        Clipboard.SetFileDropList(replacementList)
    End If

    Return returnList

End Function

' Demonstrates SetImage, ContainsImage, and GetImage.
Public Function SwapClipboardImage( _
    ByVal replacementImage As System.Drawing.Image) _
    As System.Drawing.Image

    Dim returnImage As System.Drawing.Image = Nothing

    If Clipboard.ContainsImage() Then
        returnImage = Clipboard.GetImage()
        Clipboard.SetImage(replacementImage)
    End If

    Return returnImage
End Function

' Demonstrates SetText, ContainsText, and GetText.
Public Function SwapClipboardHtmlText( _
    ByVal replacementHtmlText As String) As String

    Dim returnHtmlText As String = Nothing

    If (Clipboard.ContainsText(TextDataFormat.Html)) Then
        returnHtmlText = Clipboard.GetText(TextDataFormat.Html)
        Clipboard.SetText(replacementHtmlText, TextDataFormat.Html)
    End If

    Return returnHtmlText

End Function
```

To add data to the Clipboard in a custom format

1. Use the SetData method with a custom format name. This method is available only in .NET Framework

2.0.

You can also use predefined format names with the SetData method. For more information, see DataFormats.

```
// Demonstrates SetData, ContainsData, and GetData
// using a custom format name and a business object.
public Customer TestCustomFormat
{
    get
    {
        Clipboard.SetData("CustomerFormat", new Customer("Customer Name"));
        if (Clipboard.ContainsData("CustomerFormat"))
        {
            return Clipboard.GetData("CustomerFormat") as Customer;
        }
        return null;
    }
}
```

```
' Demonstrates SetData, ContainsData, and GetData
' using a custom format name and a business object.
Public ReadOnly Property TestCustomFormat() As Customer
    Get
        Clipboard.SetData("CustomerFormat", New Customer("Customer Name"))

        If Clipboard.ContainsData("CustomerFormat") Then
            Return CType(Clipboard.GetData("CustomerFormat"), Customer)
        End If

        Return Nothing
    End Get
End Property
```

```
[Serializable]
public class Customer
{
    private string nameValue = string.Empty;
    public Customer(String name)
    {
        nameValue = name;
    }
    public string Name
    {
        get { return nameValue; }
        set { nameValue = value; }
    }
}
```

```
<Serializable()> Public Class Customer

    Private nameValue As String = String.Empty

    Public Sub New(ByVal name As String)
        nameValue = name
    End Sub

    Public Property Name() As String
        Get
            Return nameValue
        End Get
        Set(ByVal value As String)
            nameValue = value
        End Set
    End Property

End Class
```

To add data to the Clipboard in multiple formats

1. Use the Clipboard.SetDataObject method and pass in a DataObject that contains your data. You must use this method to add data to the Clipboard on versions earlier than .NET Framework 2.0.

```
// Demonstrates how to use a DataObject to add
// data to the Clipboard in multiple formats.
public void TestClipboardMultipleFormats()
{
    DataObject data = new DataObject();

    // Add a Customer object using the type as the format.
    data.SetData(new Customer("Customer as Customer object"));

    // Add a ListViewItem object using a custom format name.
    data.SetData("CustomFormat",
        new ListViewItem("Customer as ListViewItem"));

    Clipboard.SetDataObject(data);
    DataObject retrievedData = (DataObject)Clipboard.GetDataObject();

    if (retrievedData.GetDataPresent("CustomFormat"))
    {
        ListViewItem item =
            retrievedData.GetData("CustomFormat") as ListViewItem;
        if (item != null)
        {
            MessageBox.Show(item.Text);
        }
    }

    if (retrievedData.GetDataPresent(typeof(Customer)))
    {
        Customer customer =
            retrievedData.GetData(typeof(Customer)) as Customer;
        if (customer != null)
        {
            MessageBox.Show(customer.Name);
        }
    }
}
```

```
' Demonstrates how to use a DataObject to add
' data to the Clipboard in multiple formats.
Public Sub TestClipboardMultipleFormats()

    Dim data As New DataObject()

    ' Add a Customer object using the type as the format.
    data.SetData(New Customer("Customer as Customer object"))

    ' Add a ListViewItem object using a custom format name.
    data.SetData("CustomFormat", _
        New ListViewItem("Customer as ListViewItem"))

    Clipboard.SetDataObject(data)
    Dim retrievedData As DataObject = _
        CType(Clipboard.GetDataObject(), DataObject)

    If (retrievedData.GetDataPresent("CustomFormat")) Then

        Dim item As ListViewItem = _
            TryCast(retrievedData.GetData("CustomFormat"), ListViewItem)

        If item IsNot Nothing Then
            MessageBox.Show(item.Text)
        End If

    End If

    If retrievedData.GetDataPresent(GetType(Customer)) Then

        Dim customer As Customer = _
            CType(retrievedData.GetData(GetType(Customer)), Customer)

        If customer IsNot Nothing Then

            MessageBox.Show(customer.Name)
        End If

    End If

End Sub
```

```
[Serializable]
public class Customer
{
    private string nameValue = string.Empty;
    public Customer(String name)
    {
        nameValue = name;
    }
    public string Name
    {
        get { return nameValue; }
        set { nameValue = value; }
    }
}
```

```
<Serializable()> Public Class Customer

    Private nameValue As String = String.Empty

    Public Sub New(ByVal name As String)
        nameValue = name
    End Sub

    Public Property Name() As String
        Get
            Return nameValue
        End Get
        Set(ByVal value As String)
            nameValue = value
        End Set
    End Property

End Class
```

See also

- Drag-and-Drop Operations and Clipboard Support
- How to: Retrieve Data from the Clipboard

How to: Retrieve Data from the Clipboard

11/3/2020 • 5 minutes to read • Edit Online

The Clipboard class provides methods that you can use to interact with the Windows operating system Clipboard feature. Many applications use the Clipboard as a temporary repository for data. For example, word processors use the Clipboard during cut-and-paste operations. The Clipboard is also useful for transferring information from one application to another.

Some applications store data on the Clipboard in multiple formats to increase the number of other applications that can potentially use the data. A Clipboard format is a string that identifies the format. An application that uses the identified format can retrieve the associated data on the Clipboard. The DataFormats class provides predefined format names for your use. You can also use your own format names or use an object's type as its format. For information about adding data to the Clipboard, see How to: Add Data to the Clipboard.

To determine whether the Clipboard contains data in a particular format, use one of the `Contains` *Format* methods or the GetData method. To retrieve data from the Clipboard, use one of the `Get` *Format* methods or the GetData method. These methods are new in .NET Framework 2.0.

To access data from the Clipboard by using versions earlier than .NET Framework 2.0, use the Clipboard.GetDataObject method and call the methods of the returned IDataObject. To determine whether a particular format is available in the returned object, for example, call the GetDataPresent method.

NOTE

All Windows-based applications share the system Clipboard. Therefore, the contents are subject to change when you switch to another application.

The Clipboard class can only be used in threads set to single thread apartment (STA) mode. To use this class, ensure that your `Main` method is marked with the STAThreadAttribute attribute.

To retrieve data from the Clipboard in a single, common format

1. Use the GetAudioStream, GetFileDropList, GetImage, or GetText method. Optionally, use the corresponding `Contains` *Format* methods first to determine whether data is available in a particular format. These methods are available only in .NET Framework 2.0.

```
// Demonstrates SetAudio, ContainsAudio, and GetAudioStream.
public System.IO.Stream SwapClipboardAudio(
    System.IO.Stream replacementAudioStream)
{
    System.IO.Stream returnAudioStream = null;
    if (Clipboard.ContainsAudio())
    {
        returnAudioStream = Clipboard.GetAudioStream();
        Clipboard.SetAudio(replacementAudioStream);
    }
    return returnAudioStream;
}

// Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList
public System.Collections.Specialized.StringCollection
    SwapClipboardFileDropList(
    System.Collections.Specialized.StringCollection replacementList)
{
    System.Collections.Specialized.StringCollection returnList = null;
    if (Clipboard.ContainsFileDropList())
    {
        returnList = Clipboard.GetFileDropList();
        Clipboard.SetFileDropList(replacementList);
    }
    return returnList;
}

// Demonstrates SetImage, ContainsImage, and GetImage.
public System.Drawing.Image SwapClipboardImage(
    System.Drawing.Image replacementImage)
{
    System.Drawing.Image returnImage = null;
    if (Clipboard.ContainsImage())
    {
        returnImage = Clipboard.GetImage();
        Clipboard.SetImage(replacementImage);
    }
    return returnImage;
}

// Demonstrates SetText, ContainsText, and GetText.
public String SwapClipboardHtmlText(String replacementHtmlText)
{
    String returnHtmlText = null;
    if (Clipboard.ContainsText(TextDataFormat.Html))
    {
        returnHtmlText = Clipboard.GetText(TextDataFormat.Html);
        Clipboard.SetText(replacementHtmlText, TextDataFormat.Html);
    }
    return returnHtmlText;
}
```

```
' Demonstrates SetAudio, ContainsAudio, and GetAudioStream.
Public Function SwapClipboardAudio( _
    ByVal replacementAudioStream As System.IO.Stream) _
    As System.IO.Stream

    Dim returnAudioStream As System.IO.Stream = Nothing

    If (Clipboard.ContainsAudio()) Then
        returnAudioStream = Clipboard.GetAudioStream()
        Clipboard.SetAudio(replacementAudioStream)
    End If

    Return returnAudioStream

End Function

' Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList
Public Function SwapClipboardFileDropList(ByVal replacementList _
    As System.Collections.Specialized.StringCollection) _
    As System.Collections.Specialized.StringCollection

    Dim returnList As System.Collections.Specialized.StringCollection _
        = Nothing

    If Clipboard.ContainsFileDropList() Then

        returnList = Clipboard.GetFileDropList()
        Clipboard.SetFileDropList(replacementList)
    End If

    Return returnList

End Function

' Demonstrates SetImage, ContainsImage, and GetImage.
Public Function SwapClipboardImage( _
    ByVal replacementImage As System.Drawing.Image) _
    As System.Drawing.Image

    Dim returnImage As System.Drawing.Image = Nothing

    If Clipboard.ContainsImage() Then
        returnImage = Clipboard.GetImage()
        Clipboard.SetImage(replacementImage)
    End If

    Return returnImage
End Function

' Demonstrates SetText, ContainsText, and GetText.
Public Function SwapClipboardHtmlText( _
    ByVal replacementHtmlText As String) As String

    Dim returnHtmlText As String = Nothing

    If (Clipboard.ContainsText(TextDataFormat.Html)) Then
        returnHtmlText = Clipboard.GetText(TextDataFormat.Html)
        Clipboard.SetText(replacementHtmlText, TextDataFormat.Html)
    End If

    Return returnHtmlText

End Function
```

To retrieve data from the Clipboard in a custom format

1. Use the GetData method with a custom format name. This method is available only in .NET Framework

2.0.

You can also use predefined format names with the SetData method. For more information, see DataFormats.

```
// Demonstrates SetData, ContainsData, and GetData
// using a custom format name and a business object.
public Customer TestCustomFormat
{
    get
    {
        Clipboard.SetData("CustomerFormat", new Customer("Customer Name"));
        if (Clipboard.ContainsData("CustomerFormat"))
        {
            return Clipboard.GetData("CustomerFormat") as Customer;
        }
        return null;
    }
}
```

```
' Demonstrates SetData, ContainsData, and GetData
' using a custom format name and a business object.
Public ReadOnly Property TestCustomFormat() As Customer
    Get
        Clipboard.SetData("CustomerFormat", New Customer("Customer Name"))

        If Clipboard.ContainsData("CustomerFormat") Then
            Return CType(Clipboard.GetData("CustomerFormat"), Customer)
        End If

        Return Nothing
    End Get
End Property
```

```
[Serializable]
public class Customer
{
    private string nameValue = string.Empty;
    public Customer(String name)
    {
        nameValue = name;
    }
    public string Name
    {
        get { return nameValue; }
        set { nameValue = value; }
    }
}
```

```
<Serializable()> Public Class Customer

    Private nameValue As String = String.Empty

    Public Sub New(ByVal name As String)
        nameValue = name
    End Sub

    Public Property Name() As String
        Get
            Return nameValue
        End Get
        Set(ByVal value As String)
            nameValue = value
        End Set
    End Property

End Class
```

To retrieve data from the Clipboard in multiple formats

1. Use the GetDataObject method. You must use this method to retrieve data from the Clipboard on versions earlier than .NET Framework 2.0.

```
// Demonstrates how to use a DataObject to add
// data to the Clipboard in multiple formats.
public void TestClipboardMultipleFormats()
{
    DataObject data = new DataObject();

    // Add a Customer object using the type as the format.
    data.SetData(new Customer("Customer as Customer object"));

    // Add a ListViewItem object using a custom format name.
    data.SetData("CustomFormat",
        new ListViewItem("Customer as ListViewItem"));

    Clipboard.SetDataObject(data);
    DataObject retrievedData = (DataObject)Clipboard.GetDataObject();

    if (retrievedData.GetDataPresent("CustomFormat"))
    {
        ListViewItem item =
            retrievedData.GetData("CustomFormat") as ListViewItem;
        if (item != null)
        {
            MessageBox.Show(item.Text);
        }
    }

    if (retrievedData.GetDataPresent(typeof(Customer)))
    {
        Customer customer =
            retrievedData.GetData(typeof(Customer)) as Customer;
        if (customer != null)
        {
            MessageBox.Show(customer.Name);
        }
    }
}
```

```
' Demonstrates how to use a DataObject to add
' data to the Clipboard in multiple formats.
Public Sub TestClipboardMultipleFormats()

    Dim data As New DataObject()

    ' Add a Customer object using the type as the format.
    data.SetData(New Customer("Customer as Customer object"))

    ' Add a ListViewItem object using a custom format name.
    data.SetData("CustomFormat", _
        New ListViewItem("Customer as ListViewItem"))

    Clipboard.SetDataObject(data)
    Dim retrievedData As DataObject = _
        CType(Clipboard.GetDataObject(), DataObject)

    If (retrievedData.GetDataPresent("CustomFormat")) Then

        Dim item As ListViewItem = _
            TryCast(retrievedData.GetData("CustomFormat"), ListViewItem)

        If item IsNot Nothing Then
            MessageBox.Show(item.Text)
        End If

    End If

    If retrievedData.GetDataPresent(GetType(Customer)) Then

        Dim customer As Customer = _
            CType(retrievedData.GetData(GetType(Customer)), Customer)

        If customer IsNot Nothing Then

            MessageBox.Show(customer.Name)
        End If

    End If

End Sub
```

```
[Serializable]
public class Customer
{
    private string nameValue = string.Empty;
    public Customer(String name)
    {
        nameValue = name;
    }
    public string Name
    {
        get { return nameValue; }
        set { nameValue = value; }
    }
}
```

```
<Serializable()> Public Class Customer

    Private nameValue As String = String.Empty

    Public Sub New(ByVal name As String)
        nameValue = name
    End Sub

    Public Property Name() As String
        Get
            Return nameValue
        End Get
        Set(ByVal value As String)
            nameValue = value
        End Set
    End Property

End Class
```

See also

- Drag-and-Drop Operations and Clipboard Support
- How to: Add Data to the Clipboard

Networking in Windows Forms Applications

3/9/2021 • 2 minutes to read • Edit Online

The .NET Framework provides classes for displaying Web pages, downloading Web content, interacting with file transfer protocol (FTP) sites, and consuming Web Services, making it easy to build network functionality into your application. The following resources will help you understand the networking technologies of the .NET Framework and how you can integrate them into Windows Forms.

Reference

System.Net
The root namespace for classes in the .NET Framework that handle network connectivity.

WebClient
A convenient class for retrieving Web or HTTP-based content programmatically.

FtpWebRequest
A class for retrieving and sending files with FTP.

WebBrowser
A managed wrapper class for the `WebBrowser` control that is included with Windows.

Related Sections

Network Programming in the .NET Framework
An introduction to networking in the .NET Framework.

Windows Forms Data Binding
Describes how to display database content in your application, either from a local data store or a database located on a network.

Globalizing Windows Forms applications

3/9/2021 • 2 minutes to read • Edit Online

Globalization is the process of designing and developing a software product that functions for multiple cultures.

In this section

International Fonts in Windows Forms and Controls
Explains when and how to select fonts for display of international characters on Windows Forms.

Bi-Directional Support for Windows Forms Applications
Explains how to create Windows-based applications that support bi-directional (right-to-left) languages.

Display of Asian Characters with the ImeMode Property
Introduces the `ImeMode` property, which is used to control the type of input a Windows Form or control accepts.

Related sections

- Globalizing and localizing .NET applications
- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API Using the Designer
- Localizing ClickOnce Applications
- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API
- How to: Set the Culture and UI Culture for Windows Forms Globalization
- How to: Create Mirrored Windows Forms and Controls
- How to: Support Localization on Windows Forms Using AutoSize and the TableLayoutPanel Control
- Walkthrough: Localizing Windows Forms
- Walkthrough: Creating a Layout That Adjusts Proportion for Localization

International fonts in Windows Forms and controls

11/3/2020 • 2 minutes to read • Edit Online

In International applications, the recommended method of selecting fonts is to use font fallback wherever possible. Font fallback means that the system determines what script the character belongs to.

Using font fallback

To take advantage of this feature, don't set the Font property for your form or any other element. The application will automatically use the default system font, which differs from one localized language of the operating system to another. When the application runs, the system will automatically provide the correct font for the culture selected in the operating system.

There's an exception to the rule of not setting the font, which is for changing the font style. This might be important for an application in which the user clicks a button to make text in a text box appear in boldface. To do that, you would write a function to change the text box's font style to bold, based on whatever the form's font is. It's important to call this function in two places: in the button's Click event handler and in the FontChanged event handler. If the function is called only in the Click event handler and some other piece of code changes the font family of the entire form, the text box doesn't change with the rest of the form.

```
Private Sub MakeBold()
   ' Change the TextBox to a bold version of the form font
   TextBox1.Font = New Font(Me.Font, FontStyle.Bold)
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
   ' Clicking this button makes the TextBox bold
   MakeBold()
End Sub

Private Sub Form1_FontChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.FontChanged
   ' If the TextBox is already bold and the form's font changes,
   ' change the TextBox to a bold version of the new form font
   If (TextBox1.Font.Style = FontStyle.Bold) Then
      MakeBold()
   End If
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
   // Clicking this button makes the TextBox bold
   MakeBold();
}

private void MakeBold()
{
   // Change the TextBox to a bold version of the form's font
   textBox1.Font = new Font(this.Font, FontStyle.Bold);
}

private void Form1_FontChanged(object sender, System.EventArgs e)
{
   // If the TextBox is already bold and the form's font changes,
   // change the TextBox to a bold version of the new form font
   if (textBox1.Font.Style == FontStyle.Bold)
   {
      MakeBold();
   }
}
```

However, when you localize your application, the bold font may display poorly for certain languages. If this is a concern, you want the localizers to have the option of switching the font from bold to regular text. Since localizers are typically not developers and don't have access to source code, only to resource files, this option needs to be set in the resource files. To do this, you would set the Bold property to `true`. This results in the font setting being written out to the resource files, where localizers can edit it. You then write code after the `InitializeComponent` method to reset the font based on whatever the form's font is, but using the font style specified in the resource file.

```
TextBox1.Font = New System.Drawing.Font(Me.Font, TextBox1.Font.Style)
```

```
textBox1.Font = new System.Drawing.Font(this.Font, textBox1.Font.Style);
```

See also

- Using Fonts and Text

Bi-Directional Support for Windows Forms Applications

3/9/2021 • 6 minutes to read • Edit Online

You can use Visual Studio to create Windows-based applications that support bi-directional (right-to-left) languages such as Arabic and Hebrew. This includes standard forms, dialog boxes, MDI forms, and all the controls you can work with in these forms—that is, all the objects in the Control namespace.

Culture Support

Culture and UI culture settings determine how an application works with dates, times, currency, and other information. Support for culture and UI culture is the same for bi-directional languages as it is for any other languages. For more information, see Culture-specific classes for global Windows forms and web forms.

RightToLeft and RightToLeftLayout Properties

The base Control class, from which forms derive, includes a RightToLeft property that you can set to change the reading order of a form and its controls. If you set the form's RightToLeft property, by default controls on the form inherit this setting. However, you can also set the RightToLeft property individually on most controls. Also see How to: Display Right-to-Left Text in Windows Forms for Globalization.

The effect of the RightToLeft property can differ from one control to another. In some controls the property only sets the reading order, as in the Button, TreeView and ToolTip controls. In other controls, the RightToLeft property changes both reading order and layout. This includes the RadioButton, ComboBox and CheckBox controls. Other controls require that the RightToLeftLayout property be applied to mirror its layout from right to left. The following table provides details on how the RightToLeft and RightToLeftLayout properties affect individual Windows Forms controls.

CONTROL/COMPONENT	EFFECT OF RIGHTTOLEFT PROPERTY	EFFECT OF RIGHTTOLEFTLAYOUT PROPERTY	REQUIRES MIRRORING?
Button	Sets the RTL reading order. Reverses TextAlign, ImageAlign, and TextImageRelation	No effect	No
CheckBox	The check box is displayed on the right side of the text	No effect	No
CheckedListBox	All the check boxes are displayed on the right side of the text	No effect	No
ColorDialog	Not affected; depends on the language of the operating system	No effect	No
ComboBox	Items in combo box control are right-aligned	No effect	No

CONTROL/COMPONENT	EFFECT OF RIGHTTOLEFT PROPERTY	EFFECT OF RIGHTTOLEFTLAYOUT PROPERTY	REQUIRES MIRRORING?
ContextMenu	Appears right-aligned with RTL reading order	No effect	No
DataGrid	Appears right-aligned with RTL reading order	No effect	No
DataGridView	Affects both RTL reading order and control layout	No effect	No
DateTimePicker	Not affected; depends on the language of the operating system	Mirrors the control	Yes
DomainUpDown	Left-aligns the up and down buttons	No effect	No
ErrorProvider	Not supported	No effect	No
FontDialog	Depends on the language of the operating system	No effect	No
Form	Sets RTL reading order, and reverses scrollbars	Mirrors the form	Yes
GroupBox	The caption is displayed right aligned. Child controls may inherit this property.	Use a TableLayoutPanel within the control for right-to-left mirroring support	No
HScrollBar	Starts with the scroll box (thumb) right-aligned	No effect	No
ImageList	Not required	No effect	No
Label	Displayed right-aligned. Reverses TextAlign and ImageAlign	No effect	No
LinkLabel	Displayed right-aligned. Reverses TextAlign and ImageAlign	No effect	No
ListBox	Items are right-aligned	No effect	No
ListView	Sets the reading order to RTL; elements stay left-aligned	Mirrors the control	Yes
MainMenu	Displayed right-aligned with RTL reading order at run time (not at design time)	No effect	No

CONTROL/COMPONENT	EFFECT OF RIGHTTOLEFT PROPERTY	EFFECT OF RIGHTTOLEFTLAYOUT PROPERTY	REQUIRES MIRRORING?
MaskedTextBox	Displays text from right to left.	No effect	No
MonthCalendar	Not affected; depends on the language of the operating system	Mirrors the control	Yes
NotifyIcon	Not supported	Not supported	No
NumericUpDown	Up and down buttons are left-aligned	No effect	No
OpenFileDialog	On right-to-left operating systems, setting the containing form's RightToLeft property to RightToLeft.Yes localizes the dialog	No effect	No
PageSetupDialog	Not affected; depends on the language of the operating system	No effect	No
Panel	Child controls may inherit this property	Use TableLayoutPanel within the control for right to left support	Yes
PictureBox	Not supported	No effect	No
PrintDialog	Not affected; depends on the language of the operating system	No effect	No
PrintDocument	The vertical scroll bar become left-aligned and the horizontal scroll bar starts from the left	No effect	No
PrintPreviewDialog	Not supported	Not supported	No
ProgressBar	Not affect by this property	Mirrors the control	Yes
RadioButton	The radio button is displayed on the right side of the text	No effect	No
RichTextBox	Control elements that include text are displayed from right to left with RTL reading order	No effect	No

CONTROL/COMPONENT	EFFECT OF RIGHTTOLEFT PROPERTY	EFFECT OF RIGHTTOLEFTLAYOUT PROPERTY	REQUIRES MIRRORING?
SaveFileDialog	Not affected; depends on the language of the operating system	No effect	No
SplitContainer	Panel layout is reversed; vertical scrollbar appears on the left; horizontal scrollbar starts from the right	Use a TableLayoutPanel to mirror order of child controls	No
Splitter	Not supported	No effect	No
StatusBar	Not supported; use StatusStrip instead	No effect; use StatusStrip instead	No
TabControl	Not affected by this property	Mirrors the control	Yes
TextBox	Displays text from right to left with RTL reading order	No effect	No
Timer	Not required	Not required	No
ToolBar	Not affected by this property; use ToolStrip instead	No effect; use ToolStrip instead	Yes
ToolTip	Sets the RTL reading order	No effect	No
TrackBar	The scroll or track starts from the right; when Orientation is vertical, ticks occur from the right	No effect	No
TreeView	Sets the RTL reading order only	Mirrors the control	Yes
UserControl	Vertical scrollbar appears on the left; horizontal scrollbar has thumb on the right	No direct support; use a TableLayoutPanel	No
VScrollBar	Displayed on the left side instead of right side of scrollable controls	No effect	No

Encoding

Windows Forms support Unicode, so you can include any character set when you create your bi-directional applications. However, not all Windows Forms controls support Unicode on all platforms.

GDI+

You can use GDI+ to draw text with right-to-left reading order. The DrawString method, which is used to draw text, supports a `StringFormat` parameter that you can set to the DirectionRightToLeft member of the StringFormatFlags enumeration in order to reverse the point of origin for the text.

Common Dialog Boxes

System tools such as the File Open dialog box are under the control of Windows. They inherit language elements from the operating system. If you are using a version of Windows with the correct language settings, these dialog boxes will work correctly with bi-directional languages.

Similarly, message boxes go through the operating system and support bi-directional text. The captions on message box buttons are based on the current language setting. By default, message boxes do not use right-to-left reading order, but you can specify a parameter to change the reading order when the message boxes are displayed.

RightToLeft, Scrollbars, and ScrollableControl

There is currently a limitation in Windows Forms that prevents all classes derived from ScrollableControl from acting properly when both RightToLeft is enabled and AutoScroll is set to Yes. For example, let's say that you place a control such as Panel—or a container class derived from Panel (such as FlowLayoutPanel or TableLayoutPanel)—on your form. If you set AutoScroll on the container to Yes and then set the Anchor property on one or more of the controls inside of the container to Right, then no scrollbar ever appears. The class derived from ScrollableControl acts as if AutoScroll were set to No.

Currently, the only workaround is to nest the ScrollableControl inside another ScrollableControl. For instance, if you need TableLayoutPanel to work in this situation, you can place it inside of a Panel control and set AutoScroll on the Panel to Yes.

Mirroring

Mirroring refers to reversing the layout of UI elements so that they flow from right to left. In a mirrored Windows Form, for example, the Minimize, Maximize, and Close buttons appear left-most on the title bar, not right-most.

Setting a form or control's RightToLeft property to `true` reverses the reading order of elements on a form, but this setting does not reverse the layout to be right-to-left— that is, it does not cause mirroring. For example, setting this property does not move the **Minimize**, **Maximize**, and **Close** buttons in the form's title bar to the left side of the form. Similarly, some controls, such as the TreeView control, require mirroring in order to change their display to be appropriate for Arabic or Hebrew. You can mirror these controls by settings the RightToLeftLayout property.

You can create mirrored versions of the following controls:

- ListView
- Panel
- StatusBar
- TabControl
- TabPage
- ToolBar
- TreeView

Some controls are sealed. Therefore, you cannot derive a new control from them. These include the ImageList and ProgressBar controls.

See also

- Bidirectional Support for ASP.NET Web Applications

Display of Asian Characters with the ImeMode Property

11/3/2020 • 2 minutes to read • Edit Online

The ImeMode property is used by forms and controls to force a specific mode for an input method editor (IME). The IME is an essential component for writing Chinese, Japanese, and Korean scripts, since these writing systems have more characters than can be encoded for a regular keyboard. For example, you may want to allow only ASCII characters in a particular text box. In such a case you can set the ImeMode property to ImeMode and users will only be able to enter ASCII characters for that particular text box. The default value of the ImeMode property is ImeMode, so if you set the property for a form, all controls on the form will inherit that setting. For more information, see ImeMode) and ImeMode.

See also

- Globalizing Windows Forms applications

Windows Forms and Unmanaged Applications

3/9/2021 • 2 minutes to read • Edit Online

Windows Forms applications and controls can interoperate with unmanaged applications, with some caveats. The following sections describe the scenarios and configurations that Windows Forms applications and controls support and those that they do not support.

In This Section

Windows Forms and Unmanaged Applications Overview
Offers general information about how to use and implement Windows Forms controls that work with unmanaged applications.

How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method
Provides a code example that shows how to use the Form.ShowDialog method to run a Windows Form in an unmanaged application.

How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread
Provides a code example that shows how to run a Windows Form on its own thread.

Also see Walkthrough: Supporting COM Interop by Displaying Each Windows Form on Its Own Thread.

Reference

Form.ShowDialog
Used to create a separate thread for a Windows Form.

Application.Run
Starts a message loop for a thread.

Invoke
Marshals calls from an unmanaged application to a form.

Related Sections

Exposing .NET Framework Components to COM
Offers general information about how to use .NET Framework types in unmanaged applications.

Windows Forms and Unmanaged Applications Overview

3/9/2021 • 3 minutes to read • Edit Online

Windows Forms applications and controls can interoperate with unmanaged applications, with some caveats. The following sections describe the scenarios and configurations that Windows Forms applications and controls support and those that they do not support.

Windows Forms Controls and ActiveX Applications

With the exception of Microsoft Internet Explorer and Microsoft Foundation Classes (MFC), Windows Forms controls are not supported in applications designed to host ActiveX controls. Other applications and development tools that are capable of hosting ActiveX controls, including the ActiveX test containers from versions of Visual Studio that are earlier than Visual Studio .NET 2003, are not supported hosts for Windows Forms controls.

These constraints also apply to the use of Windows Forms controls through Component Object Model COM interop. The use of a Windows Forms control through a COM callable wrapper (CCW) is supported only in Internet Explorer. For more information about COM interop, see

COM Interop.

The following table shows the available ActiveX hosting support for Windows Forms controls.

WINDOWS FORMS VERSION	SUPPORT
.NET Framework version 1.0	Internet Explorer 5.01 and later versions
.NET Framework version 1.1 and later	Internet Explorer 5.01 and later versions Microsoft Foundation Classes (MFC) 7.0 and later

Hosting Windows Forms components as ActiveX controls

In the .NET Framework 1.1, support was extended to include MFC 7.0 and later versions. This support includes any container that is fully compatible with the MFC 7.0 and later ActiveX control container.

However, registration of Windows Forms controls as ActiveX controls is not supported. Also, calling the `com.ms.win32.Ole32.CoCreateInstance` method for Windows Forms controls is not supported. Only managed activation of Windows Forms controls is supported. Once you create a Windows Forms control, you can host it in an MFC application just as with an ActiveX control.

To use Windows Forms controls in your unmanaged application, you must either host the CLR using the unmanaged CLR hosting APIs or use the C++ interop features. Using the C++ interop features is the recommended solution.

Windows Forms in COM client applications

When you open a Windows Form from a COM client application, such as a Visual Basic 6.0 application or an MFC application, the form may behave unexpectedly. For example, when you press the TAB key, the focus does not change from one control to another control. When you press the ENTER key while a command button has focus, the button's Click event is not raised. You may also experience unexpected behavior for keystrokes or mouse activity.

This behavior occurs because the unmanaged application does not implement the message loop support that Windows Forms requires to work correctly. The message loop provided by the COM client application is fundamentally different from the Windows Forms message loop.

An application's message loop is an internal program loop that retrieves messages from a thread's message queue, translates them, and then sends them to the application to be handled. The message loop for a Windows Form does not have the same architecture as message loops that earlier applications, such as Visual Basic 6.0 applications and MFC applications, provide. The window messages that are posted to the message loop may be handled differently than the Windows Form expects. Therefore, unexpected behavior may occur. Some keystroke combinations may not work, some mouse activity may not work, or some events may not be raised as expected.

Resolving Interoperability Issues

You can resolve these problems by displaying the form on a .NET Framework message loop, which is created by using the Application.Run method.

To make a Windows Form work correctly from a COM client application, you must run it on a Windows Forms message loop. To do this, use one of the following approaches:

- Use the Form.ShowDialog method to display the Windows Form. For more information, see How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method.

- Display each Windows Form on a new thread. For more information, see How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread.

See also

- Windows Forms and Unmanaged Applications
- COM Interop
- COM Interoperability in .NET Framework Applications
- COM Interoperability Samples
- Aximp.exe (Windows Forms ActiveX Control Importer)
- Exposing .NET Framework Components to COM
- Packaging an Assembly for COM
- Registering Assemblies with COM
- How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method
- How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread

How to: Support COM interop by displaying each Windows Form on its own thread

3/9/2021 • 5 minutes to read • Edit Online

You can resolve COM interoperability problems by displaying your form on a .NET Framework message loop, which you can create by using the Application.Run method.

To make a Windows Form work correctly from a COM client application, you must run the form on a Windows Forms message loop. To do this, use one of the following approaches:

- Use the Form.ShowDialog method to display the Windows Form. For more information, see How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method.
- Display each Windows Form on a separate thread.

There is extensive support for this feature in Visual Studio.

Also see Walkthrough: Supporting COM Interop by Displaying Each Windows Form on Its Own Thread.

Example

The following code example demonstrates how to display the form on a separate thread and call the Application.Run method to start a Windows Forms message pump on that thread. To use this approach, you must marshal any calls to the form from the unmanaged application by using the Invoke method.

This approach requires that each instance of a form runs on its own thread by using its own message loop. You cannot have more than one message loop running per thread. Therefore, you cannot change the client application's message loop. However, you can modify the .NET Framework component to start a new thread that uses its own message loop.

```
Imports System.Windows.Forms
Imports System.Runtime.InteropServices

<ComClass(COMForm.ClassId, COMForm.InterfaceId, COMForm.EventsId)> _
Public Class COMForm

#Region "COM GUIDs"
    ' These  GUIDs provide the COM identity for this class
    ' and its COM interfaces. If you change them, existing
    ' clients will no longer be able to access the class.
    Public Const ClassId As String = "1b49fe33-7c93-41ae-9dc7-8ac4d823286a"
    Public Const InterfaceId As String = "11651e1f-6db0-4c9e-b644-dcb79e6de2f6"
    Public Const EventsId As String = "7e61f977-b39d-47a6-8f34-f743c65ae3a3"
#End Region

    ' A creatable COM class must have a Public Sub New()
    ' with no parameters, otherwise, the class will not be
    ' registered in the COM registry and cannot be created
    ' via CreateObject.
    Public Sub New()
        MyBase.New()
    End Sub

    Private WithEvents frmManager As FormManager

    Public Sub ShowForm1()
        ' Call the StartForm method by using a new instance
        ' of the Form1 class.
        StartForm(New Form1)
    End Sub

    Private Sub StartForm(ByVal frm As Form)

        ' This procedure is used to show all forms
        ' that the client application requests. When the first form
        ' is displayed, this code will create a new message
        ' loop that runs on a new thread. The new form will
        ' be treated as the main form.

        ' Later forms will be shown on the same message loop.
        If IsNothing(frmManager) Then
            frmManager = New FormManager(frm)
        Else
            frmManager.ShowForm(frm)
        End If
    End Sub

    Private Sub frmManager_MessageLoopExit() _
    Handles frmManager.MessageLoopExit

        'Release the reference to the frmManager object.
        frmManager = Nothing

    End Sub

End Class
```

```
Imports System.Runtime.InteropServices
Imports System.Threading
Imports System.Windows.Forms

<ComVisible(False)> _
Friend Class FormManager
    ' This class is used so that you can generically pass any
    ' form that you want to the delegate.

    Private WithEvents appContext As ApplicationContext
    Private Delegate Sub FormShowDelegate(ByVal form As Form)
    Event MessageLoopExit()

    Public Sub New(ByVal MainForm As Form)
        Dim t As Thread
        If IsNothing(appContext) Then
            appContext = New ApplicationContext(MainForm)
            t = New Thread(AddressOf StartMessageLoop)
            t.IsBackground = True
            t.SetApartmentState(ApartmentState.STA)
            t.Start()
        End If
    End Sub

    Private Sub StartMessageLoop()
        ' Call the Application.Run method to run the form on its own message loop.
        Application.Run(appContext)
    End Sub

    Public Sub ShowForm(ByVal form As Form)

        Dim formShow As FormShowDelegate

        ' Start the main form first. Otherwise, focus will stay on the
        ' calling form.
        appContext.MainForm.Activate()

        ' Create a new instance of the FormShowDelegate method, and
        ' then invoke the delegate off the MainForm object.
        formShow = New FormShowDelegate( _
        AddressOf ShowFormOnMainForm_MessageLoop)

        appContext.MainForm.Invoke(formShow, New Object() {form})
    End Sub

    Private Sub ShowFormOnMainForm_MessageLoop(ByVal form As Form)
        form.Show()
    End Sub

    Private Sub ac_ThreadExit( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles appContext.ThreadExit
        appContext.MainForm.Dispose()
        appContext.MainForm = Nothing
        appContext.Dispose()
        appContext = Nothing
        RaiseEvent MessageLoopExit()
    End Sub
End Class
```

```
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form
    Private Sub Button1_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
        MessageBox.Show("Clicked button")
    End Sub

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.TextBox1 = New System.Windows.Forms.TextBox
        Me.TextBox2 = New System.Windows.Forms.TextBox
        Me.TextBox3 = New System.Windows.Forms.TextBox
        Me.Button1 = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'TextBox1
        '
        Me.TextBox1.Location = New System.Drawing.Point(12, 12)
        Me.TextBox1.Name = "TextBox1"
        Me.TextBox1.Size = New System.Drawing.Size(100, 20)
        Me.TextBox1.TabIndex = 0
        '
        'TextBox2
        '
        Me.TextBox2.Location = New System.Drawing.Point(12, 38)
        Me.TextBox2.Name = "TextBox2"
        Me.TextBox2.Size = New System.Drawing.Size(100, 20)
        Me.TextBox2.TabIndex = 1
        '
        'TextBox3
        '
        Me.TextBox3.Location = New System.Drawing.Point(12, 66)
        Me.TextBox3.Name = "TextBox3"
        Me.TextBox3.Size = New System.Drawing.Size(100, 20)
        Me.TextBox3.TabIndex = 2
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(12, 92)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(75, 23)
        Me.Button1.TabIndex = 3
        Me.Button1.Text = "Command"
        Me.Button1.UseVisualStyleBackColor = True
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(132, 146)
        Me.Controls.Add(Me.Button1)
        Me.Controls.Add(Me.TextBox3)
        Me.Controls.Add(Me.TextBox2)
        Me.Controls.Add(Me.TextBox1)
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
    Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
    Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
    Friend WithEvents Button1 As System.Windows.Forms.Button

End Class
```

Compile the code

Compile the `COMForm`, `Form1`, and `FormManager` types into an assembly called `COMWinform.dll`. Register the assembly for COM interop by using one of the methods described in Packaging an Assembly for COM. You can now use the assembly and its corresponding type library (.tlb) file in unmanaged applications. For example, you can use the type library as a reference in a Visual Basic 6.0 executable project.

See also

- Exposing .NET Framework Components to COM
- Packaging an Assembly for COM
- Registering Assemblies with COM
- How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method
- Windows Forms and Unmanaged Applications Overview

How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method

3/9/2021 • 2 minutes to read • Edit Online

You can resolve Component Object Model (COM) interoperability problems by displaying your Windows Form on a .NET Framework message loop, which is created by using the Application.Run method.

To make a form work correctly from a COM client application, you must run it on a Windows Forms message loop. To do this, use one of the following approaches:

- Use the Form.ShowDialog method to display the Windows Form;
- Display each Windows Form on a separate thread. For more information, see How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread.

Procedure

Using the Form.ShowDialog method can be the easiest way to display a form on a .NET Framework message loop because, of all the approaches, it requires the least code to implement.

The Form.ShowDialog method suspends the unmanaged application's message loop and displays the form as a dialog box. Because the host application's message loop has been suspended, the Form.ShowDialog method creates a new .NET Framework message loop to process the form's messages.

The disadvantage of using the Form.ShowDialog method is that the form will be opened as a modal dialog box. This behavior blocks any user interface (UI) in the calling application while the Windows Form is open. When the user exits the form, the .NET Framework message loop closes and the earlier application's message loop starts running again.

You can create a class library in Windows Forms which has a method to show the form, and then build the class library for COM interop. You can use this DLL file from Visual Basic 6.0 or Microsoft Foundation Classes (MFC), and from either of these environments you can call the Form.ShowDialog method to display the form.

To support COM interop by displaying a windows form with the ShowDialog method

- Replace all calls to the Form.Show method with calls to the Form.ShowDialog method in your .NET Framework component.

See also

- Exposing .NET Framework Components to COM
- How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread
- Windows Forms and Unmanaged Applications

System Information and Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

Sometimes it is necessary to gather information about the computer that your application is running on in order to make decisions in your code. For example, you might have a function that is only applicable when connected to a particular network domain; in this case you would need a way to determine the domain and disable the function if the domain is not present.

Windows Forms applications can use the SystemInformation class to determine a number of things about a computer at run time. The following example demonstrates using the SystemInformation class to retrieve the UserName and UserDomainName:

```
Dim User As String = Windows.Forms.SystemInformation.UserName
Dim Domain As String = Windows.Forms.SystemInformation.UserDomainName

MessageBox.Show("Good morning " & User & ". You are connected to " _
& Domain)
```

```
string User = SystemInformation.UserName;
string Domain = SystemInformation.UserDomainName;

MessageBox.Show("Good morning " + User + ". You are connected to "
+ Domain);
```

All members of the SystemInformation class are read-only; you cannot modify a user's settings. There are over 100 members of the class, returning information on everything from the number of monitors attached to the computer (MonitorCount) to the spacing of icons in Windows Explorer (IconHorizontalSpacing and IconVerticalSpacing).

Some of the more useful members of the SystemInformation class include ComputerName, DbcsEnabled, PowerStatus, and TerminalServerSession.

See also

- SystemInformation
- Power Management in Windows Forms

Power Management in Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

Your Windows Forms applications can take advantage of the power management features in the Windows operating system. Your applications can monitor the power status of a computer and take action when a status change occurs. For example, if your application is running on a portable computer, you might want to disable certain features in your application when the computer's battery charge falls under a certain level.

The .NET Framework provides a PowerModeChanged event that occurs whenever there is a change in power status, such as when a user suspends or resumes the operating system, or when the AC power status or battery status changes. The PowerStatus property of the SystemInformation class can be used to query for the current status, as shown in the following code example.

```
public Form1()
{
    InitializeComponent();
    SystemEvents.PowerModeChanged += new PowerModeChangedEventHandler(SystemEvents_PowerModeChanged);
}

void SystemEvents_PowerModeChanged(object sender, PowerModeChangedEventArgs e)
{
    switch (SystemInformation.PowerStatus.BatteryChargeStatus)
    {
        case System.Windows.Forms.BatteryChargeStatus.Low:
            MessageBox.Show("Battery is running low.", "Low Battery", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
            break;
        case System.Windows.Forms.BatteryChargeStatus.Critical:
            MessageBox.Show("Battery is critcally low.", "Critical Battery", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
            break;
        default:
            // Battery is okay.
            break;
    }
}
```

```
Public Sub New()
    InitializeComponent()
    AddHandler Microsoft.Win32.SystemEvents.PowerModeChanged, AddressOf PowerModeChanged
End Sub

Private Sub PowerModeChanged(ByVal Sender As System.Object, ByVal e As
Microsoft.Win32.PowerModeChangedEventArgs)
    Select Case SystemInformation.PowerStatus.BatteryChargeStatus
        Case BatteryChargeStatus.Low
            MessageBox.Show("Battery is running low.", "Low Battery", MessageBoxButtons.OK, _
                            System.Windows.Forms.MessageBoxIcon.Exclamation)
        Case BatteryChargeStatus.Critical
            MessageBox.Show("Battery is critically low.", "Critical Battery", MessageBoxButtons.OK, _
                            System.Windows.Forms.MessageBoxIcon.Stop)
        Case Else
            ' Battery is okay.
            Exit Select
    End Select
End Sub
```

Besides the BatteryChargeStatus enumerations, the PowerStatus property also contains enumerations for determining battery capacity (BatteryFullLifetime) and battery charge percentage (BatteryLifePercent, BatteryLifeRemaining).

You can use the SetSuspendState method of the Application to put a computer into hibernation or suspend mode. If the `force` argument is set to `false`, the operating system will broadcast an event to all applications requesting permission to suspend. If the `disableWakeEvent` argument is set to `true`, the operating system disables all wake events.

The following code example demonstrates how to put a computer into hibernation.

```
if (SystemInformation.PowerStatus.BatteryChargeStatus == System.Windows.Forms.BatteryChargeStatus.Critical)
{
    Application.SetSuspendState(PowerState.Hibernate, false, false);
}
```

```
If SystemInformation.PowerStatus.BatteryChargeStatus = System.Windows.Forms.BatteryChargeStatus.Critical
Then
    Application.SetSuspendState(PowerState.Hibernate, False, False)
End If
```

See also

- PowerModeChanged
- PowerStatus
- SetSuspendState
- SessionSwitch

Help Systems in Windows Forms Applications

11/3/2020 • 2 minutes to read • Edit Online

One of the most important courtesies you, as a developer of applications, can furnish your users with is a competent Help system. This is where they will turn when they become confused or disoriented. Providing a Help system in a Windows-based application is easily done by using the HelpProvider Component.

Different Types of Help

The Windows Forms HelpProvider component is used to associate an HTML Help 1.x Help file (either a .chm file, produced with the HTML Help Workshop, or an .htm file) with your Windows-based application. The HelpProvider component can be used to provide context-sensitive Help for controls on Windows Forms or specific controls. Additionally, the HelpProvider component can open a Help file to specific areas, such as the main page of a table of contents, an index, or a search function. For general information about the HelpProvider component, see HelpProvider Component Overview. For information on how to use the HelpProvider component to show pop-up Help on Windows Forms, see How to: Display Pop-up Help. For information on using the ToolTip component to show control-specific Help, see Control Help Using ToolTips.

You can generate HTML Help 1.x files with the HTML Help Workshop. For more information on HTML Help, see the "HTML Help Workshop" or the other "HTML Help" topics in MSDN.

See also

- Integrating User Help in Windows Forms
- HelpProvider Component
- ToolTip Component
- Windows Forms Overview
- Windows Forms

Windows Forms Visual Inheritance

3/9/2021 • 2 minutes to read • Edit Online

Occasionally, you may decide that a project calls for a form similar to one that you have created in a previous project. Or, you may want to create a basic form with settings such as a watermark or certain control layout that you will then use again within a project, with each iteration containing modifications to the original form template. Form inheritance enables you to create a base form and then inherit from it and make modifications while preserving whatever original settings you need.

You can create derived-class forms programmatically or by using the Visual Inheritance picker.

In This Section

How to: Inherit Windows Forms
Gives directions for creating inherited forms in code.

How to: Inherit Forms Using the Inheritance Picker Dialog Box
Gives directions for creating inherited forms with the Inheritance Picker.

Effects of Modifying a Base Form's Appearance
Gives directions for changing a base form's controls and their properties.

Walkthrough: Demonstrating Visual Inheritance
Describes how to create a base Windows Form and compile it into a class library. You will import this class library into another project, and create a new form that inherits from the base form.

How to: Use the Modifiers and GenerateMember Properties
Gives directions for using the `GenerateMember` and `Modifiers` properties, which are relevant when the Windows Forms Designer generates a member variable for a component.

Related Sections

Inheritance basics (Visual Basic)
Describes how to define Visual Basic classes that serve as the basis for other classes.

class
Describes the C# approach of classes, in which single inheritance is allowed.

Troubleshooting Inherited Event Handlers in Visual Basic
Lists common issues that arise with event handlers in inherited components

How to: Inherit Windows Forms

3/9/2021 • 2 minutes to read • Edit Online

Creating new Windows Forms by inheriting from base forms is a handy way to duplicate your best efforts without going through the process of entirely recreating a form every time you require it.

For more information about inheriting forms at design time using the **Inheritance Picker** dialog box and how to visually distinguish between security levels of inherited controls, see How to: Inherit Forms Using the Inheritance Picker Dialog Box.

NOTE

In order to inherit from a form, the file or namespace containing that form must have been built into an executable file or DLL. To build the project, choose **Build** from the **Build** menu. Also, a reference to the namespace must be added to the class inheriting the form.

Inherit a form programmatically

1. In your class, add a reference to the namespace containing the form you wish to inherit from.

2. In the class definition, add a reference to the form to inherit from. The reference should include the namespace that contains the form, followed by a period, then the name of the base form itself.

```
Public Class Form2
    Inherits Namespace1.Form1
```

```
public class Form2 : Namespace1.Form1
```

When inheriting forms, keep in mind that issues may arise with regard to event handlers being called twice, because each event is being handled by both the base class and the inherited class. For more information on how to avoid this problem, see Troubleshooting Inherited Event Handlers in Visual Basic.

See also

- Inherits Statement
- Imports Statement (.NET Namespace and Type)
- using
- Effects of Modifying a Base Form's Appearance
- Windows Forms Visual Inheritance

How to: Inherit Forms Using the Inheritance Picker

3/9/2021 • 3 minutes to read • Edit Online

The easiest way to inherit a form or other object is to use the **Inheritance Picker** dialog box. With it, you can take advantage of code or user interfaces (UI) you have already created in other solutions.

NOTE

In order to inherit from a form with the **Inheritance Picker** dialog box, the project containing that form must have been built into an executable file or DLL. To build the project, choose **Build Solution** from the **Build** menu.

Create a Windows Form by using the Inheritance Picker

1. In Visual Studio, from the **Project** menu, choose **Add Windows Form**.

   The **Add New Item** dialog box opens.

2. Search the **Inherited Form** template either from the searchbox or by clicking on the **Windows Forms** category, select it, and name it in the **Name** box. Click the **Add** button to proceed.

   The **Inheritance Picker** dialog box opens. If the current project already contains forms, they are displayed in the **Inheritance Picker** dialog box.

3. To inherit from a form in another assembly, click the **Browse** button.

4. Within the **Select a file which contains a component to inherit from** dialog box, navigate to the project containing the form or module you desire.

5. Click the name of the .exe or .dll file to select it and click the **Open** button.

   This returns you to the **Inheritance Picker** dialog box, where the component is now listed, along with the project in which it is located.

6. Select the component.

   In **Solution Explorer**, the component is added to your project. If it has a UI, controls that are part of the inherited form will be marked with a glyph (), and, when selected, have a border indicating the level of security that the control has on the superclassed form. The behaviors that correspond to the different security levels are listed in the table below.

SECURITY LEVEL OF CONTROL	AVAILABLE INTERACTION THROUGH DESIGNER AND CODE EDITOR WITH INHERITED FORM
Public	Standard border with sizing handles: control may be sized and moved. The control can be accessed internally by the class which declares it and externally by other classes.
Protected	Standard border with sizing handles: control may be sized and moved. Can be accessed internally by the class that declares it and any class that inherits from the parent class, but cannot be accessed by external classes.

SECURITY LEVEL OF CONTROL	AVAILABLE INTERACTION THROUGH DESIGNER AND CODE EDITOR WITH INHERITED FORM
Protected Internal (Protected Friend in Visual Basic)	Standard border with sizing handles: control may be sized and moved. Can be accessed internally by the class that declares it, by any class that inherits from the parent class, and by other members of the assembly that contains it.
Internal (Friend in Visual Basic)	Standard border with no sizing handles, shown on the form, properties visible in **Properties** window. However, all aspects of the control will be considered read-only. You cannot move or size the control, or change its properties. If the control is a container of other controls, like a group box, new controls cannot be added and existing controls cannot be removed, even if those controls were public. The control can only be accessed by other members of the assembly that contains it.
Private	Standard border with no sizing handles, shown on the form, properties visible in **Properties** window. However, all aspects of the control will be considered read-only. You cannot move or size the control, or change its properties. If the control is a container of other controls, like a group box, new controls cannot be added and existing controls cannot be removed, even if those controls were public. The control can only be accessed by the class that declares it.

For information about how to alter a base form's appearance, see Effects of Modifying a Base Form's Appearance.

NOTE

When you combine inherited controls and components with standard controls and components on Windows Forms, you might encounter conflicts with the z-ordering. You can correct this by modifying the z-order, which is done by clicking in the **Format** menu, pointing to **Order**, and then clicking **Bring To Front** or **Send To Back**. For more information about the z-order of controls, see How to: Layer Objects on Windows Forms.

See also

- Inherits Statement
- using
- Effects of Modifying a Base Form's Appearance
- Windows Forms Visual Inheritance

Effects of modifying a base form's appearance

3/9/2021 • 2 minutes to read • Edit Online

During application development, you may often need to change the appearance of the base form from which other forms in the project (or in other projects) are inheriting.

At design time, changes to the base form's appearance (be it the setting of properties or the addition and subtraction of controls) are reflected on inherited forms when the project containing the base form is built. It is not sufficient for you to simply save the changes to the base form. To build a project, choose **Build** from the **Build** menu.

Modifications made to the base form at run time have no affect on inherited forms that are already instantiated.

See also

- base
- How to: Inherit Windows Forms
- Windows Forms Visual Inheritance

Walkthrough: Demonstrating Visual Inheritance

11/3/2020 • 4 minutes to read • Edit Online

Visual inheritance enables you to see the controls on the base form and to add new controls. In this walkthrough you will create a base form and compile it into a class library. You will import this class library into another project and create a new form that inherits from the base form. During this walkthrough, you will learn how to:

- Create a class library project containing a base form.
- Add a button with properties that derived classes of the base form can modify.
- Add a button that cannot be modified by inheritors of the base form.
- Create a project containing a form that inherits from `BaseForm`.

Ultimately, this walkthrough will demonstrate the difference between private and protected controls on an inherited form.

Caution

Not all controls support visual inheritance from a base form. The following controls do not support the scenario described in this walkthrough:

- WebBrowser
- ToolStrip
- ToolStripPanel
- TableLayoutPanel
- FlowLayoutPanel
- DataGridView

These controls in the inherited form are always read-only regardless of the modifiers you use (`private`, `protected`, or `public`).

Create a class library project containing a base form

1. In Visual Studio, from the **File** menu, choose **New** > **Project** to open the **New Project** dialog box.
2. Create a Windows Forms application named `BaseFormLibrary`.
3. To create a class library instead of a standard Windows Forms application, in **Solution Explorer**, right-click the **BaseFormLibrary** project node and then select **Properties**.
4. In the properties for the project, change the **Output type** from **Windows Application** to **Class Library**.
5. From the **File** menu, choose **Save All** to save the project and files to the default location.

The next two procedures add buttons to the base form. To demonstrate visual inheritance, you will give the buttons different access levels by setting their `Modifiers` properties.

Add a button that inheritors of the base form can modify

1. Open **Form1** in the designer.

2. On the **All Windows Forms** tab of the **Toolbox**, double-click **Button** to add a button to the form. Use the mouse to position and resize the button.

3. In the Properties window, set the following properties of the button:

   - Set the **Text** property to **Say Hello**.
   - Set the **(Name)** property to **btnProtected**.
   - Set the **Modifiers** property to **Protected**. This makes it possible for forms that inherit from **Form1** to modify the properties of **btnProtected**.

4. Double-click the **Say Hello** button to add an event handler for the **Click** event.

5. Add the following line of code to the event handler:

```
MessageBox.Show("Hello, World!")
```

```
MessageBox.Show("Hello, World!");
```

Add a button that cannot be modified by inheritors of the base form

1. Switch to design view by clicking the **Form1.vb [Design], Form1.cs [Design], or Form1.jsl [Design]** tab above the code editor, or by pressing F7.

2. Add a second button and set its properties as follows:

   - Set the **Text** property to **Say Goodbye**.
   - Set the **(Name)** property to **btnPrivate**.
   - Set the **Modifiers** property to **Private**. This makes it impossible for forms that inherit from **Form1** to modify the properties of **btnPrivate**.

3. Double-click the **Say Goodbye** button to add an event handler for the **Click** event. Place the following line of code in the event procedure:

```
MessageBox.Show("Goodbye!")
```

```
MessageBox.Show("Goodbye!");
```

4. From the **Build** menu, choose **Build BaseForm Library** to build the class library.

   Once the library is built, you can create a new project that inherits from the form you just created.

Create a project containing a form that inherits from the base form

1. From the **File** menu, choose **Add** and then **New Project** to open the **Add New Project** dialog box.

2. Create a Windows Forms application named `InheritanceTest`.

Add an inherited form

1. In **Solution Explorer**, right-click the **InheritanceTest** project, select **Add**, and then select **New Item**.

2. In the **Add New Item** dialog box, select the **Windows Forms** category (if you have a list of categories)

and then select the **Inherited Form** template.

3. Leave the default name of `Form2` and then click **Add**.

4. In the **Inheritance Picker** dialog box, select **Form1** from the **BaseFormLibrary** project as the form to inherit from and click **OK**.

   This creates a form in the **InheritanceTest** project that derives from the form in **BaseFormLibrary**.

5. Open the inherited form (**Form2**) in the designer by double-clicking it, if it is not already open.

   In the designer, the inherited buttons have a symbol (⧉) in their upper corner, indicating they are inherited.

6. Select the **Say Hello** button and observe the resize handles. Because this button is protected, the inheritors can move it, resize it, change its caption, and make other modifications.

7. Select the private **Say Goodbye** button, and notice that it does not have resize handles. Additionally, in the **Properties** window, the properties of this button are grayed to indicate they cannot be modified.

8. If you are using Visual C#:

   a. In **Solution Explorer**, right-click **Form1** in the **InheritanceTest** project and then choose **Delete**. In the message box that appears, click **OK** to confirm the deletion.

   b. Open the Program.cs file and change the line `Application.Run(new Form1());` to `Application.Run(new Form2());`.

9. In **Solution Explorer**, right-click the **InheritanceTest** project and select **Set As Startup Project**.

10. In **Solution Explorer**, right-click the **InheritanceTest** project and select **Properties**.

11. In the **InheritanceTest** property pages, set the **Startup object** to be the inherited form (**Form2**).

12. Press **F5** to run the application, and observe the behavior of the inherited form.

Next steps

Inheritance for user controls works in much the same way. Open a new class library project and add a user control. Place constituent controls on it and compile the project. Open another new class library project and add a reference to the compiled class library. Also, try adding an inherited control (through the **Add New Items** dialog box) to the project and using the **Inheritance Picker**. Add a user control, and change the `Inherits` (`:` in Visual C#) statement. For more information, see How to: Inherit Windows Forms.

See also

- How to: Inherit Windows Forms
- Windows Forms Visual Inheritance
- Windows Forms

How to: Use the Modifiers and GenerateMember Properties

11/3/2020 • 2 minutes to read • Edit Online

When you place a component on a Windows Form, two properties are provided by the design environment: `GenerateMember` and `Modifiers`. The `GenerateMember` property specifies when the Windows Forms Designer generates a member variable for a component. The `Modifiers` property is the access modifier assigned to that member variable. If the value of the `GenerateMember` property is `false`, the value of the `Modifiers` property has no effect.

Specify whether a component is a member of the form

1. In Visual Studio, in the Windows Forms Designer, open your form.

2. Open the **Toolbox**, and on the form, place three Button controls.

3. Set the `GenerateMember` and `Modifiers` properties for each Button control according to the following table.

BUTTON NAME	GENERATEMEMBER VALUE	MODIFIERS VALUE
`button1`	`true`	`private`
`button2`	`true`	`protected`
`button3`	`false`	No change

4. Build the solution.

5. In **Solution Explorer**, click the **Show All Files** button.

6. Open the **Form1** node, and in the **Code Editor**,open the **Form1.Designer.vb** or **Form1.Designer.cs** file. This file contains the code emitted by the Windows Forms Designer.

7. Find the declarations for the three buttons. The following code example shows the differences specified by the `GenerateMember` and `Modifiers` properties.

```
private void InitializeComponent()
{
    // button3 is declared in a local scope, because
    // its GenerateMember property is false.
    System.Windows.Forms.Button button3;
    this.button1 = new System.Windows.Forms.Button();
    this.button2 = new System.Windows.Forms.Button();
    button3 = new System.Windows.Forms.Button();
```

```
Private Sub InitializeComponent()

    ' button3 is declared in a local scope, because
    ' its GenerateMember property is false.
    Dim button3 As System.Windows.Forms.Button
    Me.button1 = New System.Windows.Forms.Button()
    Me.button2 = New System.Windows.Forms.Button()
    button3 = New System.Windows.Forms.Button()
```

```
// The Modifiers property for button1 is "private".
private Button button1;

// The Modifiers property for button2 is "protected".
protected Button button2;

// button3 is not a member, because
// its GenerateMember property is false.
```

```
 ' The Modifiers property for button1 is "Private".
 Private button1 As Button

 ' The Modifiers property for button2 is "Protected".
 Protected button2 As Button

' button3 is not a member, because
' its GenerateMember property is false.
```

NOTE

By default, the Windows Forms Designer assigns the `private` (`Friend` in Visual Basic) modifier to container controls like Panel. If your base UserControl or Form has a container control, it will not accept new children in inherited controls and forms. The solution is to change the modifier of the base container control to `protected` or `public`.

See also

- Button
- Windows Forms Visual Inheritance
- Walkthrough: Demonstrating Visual Inheritance
- How to: Inherit Windows Forms

Multiple-Document Interface (MDI) Applications

11/3/2020 • 2 minutes to read • Edit Online

Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window. MDI applications often have a Window menu item with submenus for switching between windows or documents.

NOTE

There are some behavior differences between MDI forms and single-document interface (SDI) windows in Windows Forms. The `Opacity` property does not affect the appearance of MDI child forms. Additionally, the CenterToParent method does not affect the behavior of MDI child forms.

In This Section

How to: Create MDI Parent Forms
Gives directions for creating the container for the multiple documents within an MDI application.

How to: Create MDI Child Forms
Gives directions for creating one or more windows that operate within an MDI parent form.

How to: Determine the Active MDI Child
Gives directions for verifying the child window that has focus (and sending its contents to the Clipboard).

How to: Send Data to the Active MDI Child
Gives directions for transporting information to the active child window.

How to: Arrange MDI Child Forms
Gives directions for tiling, cascading, or arranging the child windows of an MDI application.

How to: Create MDI Parent Forms

11/3/2020 • 2 minutes to read • Edit Online

IMPORTANT

This topic uses the MainMenu control, which has been replaced by the MenuStrip control. The MainMenu control is retained for both backward compatibility and future use, if you choose. For information about creating a MDI parent Form by using a MenuStrip, see How to: Create an MDI Window List with MenuStrip.

The foundation of a Multiple-Document Interface (MDI) application is the MDI parent form. This is the form that contains the MDI child windows, which are the sub-windows wherein the user interacts with the MDI application. Creating an MDI parent form is easy, both in the Windows Forms Designer and programmatically.

Create an MDI parent form at design time

1. Create a Windows Application project in Visual Studio.
2. In the **Properties** window, set the IsMdiContainer property to **true**.

   This designates the form as an MDI container for child windows.

   NOTE

   While setting properties in the **Properties** window, you can also set the `WindowState` property to **Maximized**, if you like, as it is easiest to manipulate MDI child windows when the parent form is maximized. Additionally, be aware that the edge of the MDI parent form will pick up the system color (set in the Windows System Control Panel), rather than the back color you set using the Control.BackColor property.

3. From the **Toolbox**, drag a **MenuStrip** control to the form. Create a top-level menu item with the **Text** property set to **&File** with submenu items called **&New** and **&Close**. Also create a top-level menu item called **&Window**.

   The first menu will create and hide menu items at run time, and the second menu will keep track of the open MDI child windows. At this point, you have created an MDI parent window.

4. Press **F5** to run the application. For information about creating MDI child windows that operate within the MDI parent form, see How to: Create MDI Child Forms.

See also

- Multiple-Document Interface (MDI) Applications
- How to: Create MDI Child Forms
- How to: Determine the Active MDI Child
- How to: Send Data to the Active MDI Child
- How to: Arrange MDI Child Forms

How to: Create MDI child forms

11/3/2020 • 4 minutes to read • Edit Online

MDI child forms are an essential element of Multiple-Document Interface (MDI) applications, as these forms are the center of user interaction.

In the following procedure, you'll use Visual Studio to create an MDI child form that displays a RichTextBox control, similar to most word-processing applications. By substituting the System.Windows.Forms control with other controls, such as the DataGridView control, or a mixture of controls, you can create MDI child windows (and, by extension, MDI applications) with diverse possibilities.

Create MDI child forms

1. Create a new Windows Forms application project in Visual Studio. In the **Properties** window for the form, set its IsMdiContainer property to `true` and its `WindowsState` property to `Maximized`.

   This designates the form as an MDI container for child windows.

2. From the `Toolbox`, drag a MenuStrip control to the form. Set its `Text` property to **File**.

3. Click the ellipsis (…) next to the **Items** property, and click **Add** to add two child tool strip menu items. Set the `Text` property for these items to **New** and **Window**.

4. In **Solution Explorer**, right-click the project, and then select **Add** > **New Item**.

5. In the **Add New Item** dialog box, select **Windows Form** (in Visual Basic or in Visual C#) or **Windows Forms Application (.NET)** (in Visual C++) from the **Templates** pane. In the **Name** box, name the form **Form2**. Select **Open** to add the form to the project.

   NOTE

   The MDI child form you created in this step is a standard Windows Form. As such, it has an Opacity property, which enables you to control the transparency of the form. However, the Opacity property was designed for top-level windows. Do not use it with MDI child forms, as painting problems can occur.

   This form will be the template for your MDI child forms.

   The **Windows Forms Designer** opens, displaying **Form2**.

6. From the **Toolbox**, drag a **RichTextBox** control to the form.

7. In the **Properties** window, set the `Anchor` property to **Top, Left** and the `Dock` property to **Fill**.

   This causes the RichTextBox control to completely fill the area of the MDI child form, even when the form is resized.

8. Double click the **New** menu item to create a Click event handler for it.

9. Insert code similar to the following to create a new MDI child form when the user clicks the **New** menu item.

NOTE

In the following example, the event handler handles the Click event for `MenuItem2`. Be aware that, depending on the specifics of your application architecture, your **New** menu item may not be `MenuItem2`.

```
Protected Sub MDIChildNew_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MenuItem2.Click
   Dim NewMDIChild As New Form2()
   'Set the Parent Form of the Child window.
   NewMDIChild.MdiParent = Me
   'Display the new form.
   NewMDIChild.Show()
End Sub
```

```
protected void MDIChildNew_Click(object sender, System.EventArgs e){
   Form2 newMDIChild = new Form2();
   // Set the Parent Form of the Child window.
   newMDIChild.MdiParent = this;
   // Display the new form.
   newMDIChild.Show();
}
```

```
private:
   void menuItem2_Click(System::Object ^ sender,
      System::EventArgs ^ e)
   {
      Form2^ newMDIChild = gcnew Form2();
      // Set the Parent Form of the Child window.
      newMDIChild->MdiParent = this;
      // Display the new form.
      newMDIChild->Show();
   }
```

In C++, add the following `#include` directive at the top of Form1.h:

```
#include "Form2.h"
```

10. In the drop-down list at the top of the **Properties** window, select the menu strip that corresponds to the **File** menu strip and set the MdiWindowListItem property to the Window ToolStripMenuItem.

    This enables the **Window** menu to maintain a list of open MDI child windows with a check mark next to the active child window.

11. Press **F5** to run the application. By selecting **New** from the **File** menu, you can create new MDI child forms, which are kept track of in the **Window** menu item.

NOTE

When an MDI child form has a MainMenu component (with, usually, a menu structure of menu items) and it is opened within an MDI parent form that has a MainMenu component (with, usually, a menu structure of menu items), the menu items will merge automatically if you have set the MergeType property (and optionally, the MergeOrder property). Set the MergeType property of both MainMenu components and all of the menu items of the child form to MergeItems. Additionally, set the MergeOrder property so that the menu items from both menus appear in the desired order. Moreover, keep in mind that when you close an MDI parent form, each of the MDI child forms raises a Closing event before the Closing event for the MDI parent is raised. Canceling an MDI child's Closing event will not prevent the MDI parent's Closing event from being raised; however, the CancelEventArgs argument for the MDI parent's Closing event will now be set to `true`. You can force the MDI parent and all MDI child forms to close by setting the CancelEventArgs argument to `false`.

See also

- Multiple-Document Interface (MDI) Applications
- How to: Create MDI Parent Forms
- How to: Determine the Active MDI Child
- How to: Send Data to the Active MDI Child
- How to: Arrange MDI Child Forms

How to: Determine the Active MDI Child

11/3/2020 • 2 minutes to read • Edit Online

On occasion, you will want to provide a command that operates on the control that has focus on the currently active child form. For example, suppose you want to copy selected text from the child form's text box to the Clipboard. You would create a procedure that copies selected text to the Clipboard using the Click event of the Copy menu item on the standard Edit menu.

Because an MDI application can have many instances of the same child form, the procedure needs to know which form to use. To specify the correct form, use the ActiveMdiChild property, which returns the child form that has the focus or that was most recently active.

When you have several controls on a form, you also need to specify which control is active. Like the ActiveMdiChild property, the ActiveControl property returns the control with the focus on the active child form. The procedure below illustrates a copy procedure that can be called from a child form menu, a menu on the MDI form, or a toolbar button.

To determine the active MDI child (to copy its text to the Clipboard)

1. Within a method, copy the text of the active control of the active child form to the Clipboard.

   NOTE

   This example assumes there is an MDI parent form (`Form1`) that has one or more MDI child windows containing a RichTextBox control. For more information, see Creating MDI Parent Forms.

```
Public Sub mniCopy_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles mniCopy.Click

   ' Determine the active child form.
   Dim activeChild As Form = Me.ActiveMDIChild

   ' If there is an active child form, find the active control, which
   ' in this example should be a RichTextBox.
   If (Not activeChild Is Nothing) Then
      Dim theBox As RichTextBox = _
        TryCast(activeChild.ActiveControl, RichTextBox)

      If (Not theBox Is Nothing) Then
         'Put selected text on Clipboard.
         Clipboard.SetDataObject(theBox.SelectedText)
      Else
         MessageBox.Show("You need to select a RichTextBox.")
      End If
   End If
End Sub
```

```
protected void mniCopy_Click (object sender, System.EventArgs e)
{
   // Determine the active child form.
   Form activeChild = this.ActiveMdiChild;

   // If there is an active child form, find the active control, which
   // in this example should be a RichTextBox.
   if (activeChild != null)
   {
      try
      {
         RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;
         if (theBox != null)
         {
            // Put the selected text on the Clipboard.
            Clipboard.SetDataObject(theBox.SelectedText);

         }
      }
      catch
      {
         MessageBox.Show("You need to select a RichTextBox.");
      }
   }
}
```

See also

- Multiple-Document Interface (MDI) Applications
- How to: Create MDI Parent Forms
- How to: Create MDI Child Forms
- How to: Send Data to the Active MDI Child
- How to: Arrange MDI Child Forms

How to: Send Data to the Active MDI Child

11/3/2020 • 2 minutes to read • Edit Online

Often, within the context of Multiple-Document Interface (MDI) Applications, you will need to send data to the active child window, such as when the user pastes data from the Clipboard into an MDI application.

NOTE

For information about verifying which child window has focus and sending its contents to the Clipboard, see Determining the Active MDI Child.

To send data to the active MDI child window from the Clipboard

1. Within a method, copy the text on the Clipboard to the active control of the active child form.

   NOTE

   This example assumes there is an MDI parent form (`Form1`) that has one or more MDI child windows containing a RichTextBox control. For more information, see Creating MDI Parent Forms.

```
Public Sub mniPaste_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles mniPaste.Click

   ' Determine the active child form.
   Dim activeChild As Form = Me.ParentForm.ActiveMDIChild

   ' If there is an active child form, find the active control, which
   ' in this example should be a RichTextBox.
   If (Not activeChild Is Nothing) Then
      Try
         Dim theBox As RichTextBox = Ctype(activeChild.ActiveControl, RichTextBox)
         If (Not theBox Is Nothing) Then
            ' Create a new instance of the DataObject interface.
            Dim data As IDataObject = Clipboard.GetDataObject()
            ' If the data is text, then set the text of the
            ' RichTextBox to the text in the clipboard.
            If (data.GetDataPresent(DataFormats.Text)) Then
               theBox.SelectedText = data.GetData(DataFormats.Text).ToString()
            End If
         End If
      Catch
         MessageBox.Show("You need to select a RichTextBox.")
      End Try
   End If
End Sub
```

```
protected void mniPaste_Click (object sender, System.EventArgs e)
{
  // Determine the active child form.
   Form activeChild = this.ParentForm.ActiveMdiChild;

   // If there is an active child form, find the active control, which
   // in this example should be a RichTextBox.
   if (activeChild != null)
   {
      try
      {
         RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;
         if (theBox != null)
         {
            // Create a new instance of the DataObject interface.
            IDataObject data = Clipboard.GetDataObject();
            // If the data is text, then set the text of the
            // RichTextBox to the text in the clipboard.
            if (data.GetDataPresent(DataFormats.Text))
            {
               theBox.SelectedText = data.GetData(DataFormats.Text).ToString();
            }
         }
      }
      catch
      {
         MessageBox.Show("You need to select a RichTextBox.");
      }
   }
}
```

See also

- Multiple-Document Interface (MDI) Applications
- How to: Create MDI Parent Forms
- How to: Create MDI Child Forms
- How to: Determine the Active MDI Child
- How to: Arrange MDI Child Forms

How to: Arrange MDI Child Forms

11/3/2020 • 2 minutes to read • Edit Online

Often, applications will have menu commands for actions such as Tile, Cascade, and Arrange, which control the layout of the open MDI child forms. You can use the LayoutMdi method with one of the MdiLayout enumeration values to rearrange the child forms in an MDI parent form.

The MdiLayout enumeration values display child forms as cascading, as horizontally or vertically tiled, or as child form icons arranged along the lower portion of the MDI form. These values have the same effect as the Windows commands **Cascade windows**, **Show windows side by side**, **Show windows stacked**, and **Show the desktop**, respectively.

Often, these methods are used as the event handlers called by a menu item's Click event. In this way, a menu item with the text "Cascade Windows" can have the desired effect on the MDI child windows.

To arrange child forms

1. In a method, use the LayoutMdi method to set the MdiLayout enumeration for the MDI parent form. The following example uses the MdiLayout.Cascade enumeration value for the child windows of the MDI parent form (`Form1`). The enumeration is used in code during the event handler for the Click event of the **Cascade Windows** menu item.

```
Protected Sub CascadeWindows_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
   Me.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade)
End Sub
```

```
protected void CascadeWindows_Click(object sender, System.EventArgs e){
   this.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade);
}
```

> **NOTE**
>
> You can also tile windows and arranging windows as icons by changing the MdiLayout enumeration value used.

2. If you're using Visual C#, place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

See also

- Multiple-Document Interface (MDI) Applications
- How to: Create MDI Parent Forms
- How to: Create MDI Child Forms
- How to: Determine the Active MDI Child
- How to: Send Data to the Active MDI Child

Integrating User Help in Windows Forms

11/3/2020 • 2 minutes to read • Edit Online

An essential, but often overlooked, aspect of building Windows-based applications is the Help system, as this is where users turn for assistance in times of confusion. Windows Forms support two different types of Help, each provided by the HelpProvider Component. The first involves pointing the user to a Help file of either HTML or HTML Help 1.*x* or greater format. The second can display brief "What's This"-type Help on individual controls; this is especially useful on dialog boxes. Both types of Help can be used on the same form.

Additionally, the ToolTip Component can be used to provide individual Help for controls on Windows Forms.

In This Section

How to: Provide Help in a Windows Application
Explains how to use the `HelpProvider` component to link controls to files in a Help system.

How to: Display Pop-up Help
Explains how to use the `HelpProvider` component to show pop-up Help on Windows Forms.

Control Help Using ToolTips
Describes using the `ToolTip` component to show control-specific Help.

Related Sections

HelpProvider Component
Explains the basics of the `HelpProvider` component.

ToolTip Component
Explains the basics of the `ToolTip` component.

Windows Forms Overview
Explains the basics of Windows Forms.

Windows Forms
Provides an overview of Windows Forms.

How to: Provide Help in a Windows Application

11/3/2020 • 2 minutes to read • Edit Online

You can make use of the HelpProvider component to attach Help topics within a Help file to specific controls on Windows Forms. The Help file can be either HTML or HTMLHelp 1.x or greater format.

Provide Help

1. In Visual Studio, from the **Toolbox**, drag a HelpProvider component to your form.

   The component will reside in the tray at the bottom of the Windows Forms Designer.

2. In the **Properties** window, set the HelpNamespace property to the .chm, .col, or .htm Help file.

3. Select another control you have on your form, and in the **Properties** window, set the SetHelpKeyword property.

   This is the string passed through the HelpProvider component to your Help file to summon the appropriate Help topic.

4. In the **Properties** window, set the SetHelpNavigator property to a value of the HelpNavigator enumeration.

   This determines the way in which the **HelpKeyword** property is passed to the Help system. The following table shows the possible settings and their descriptions.

MEMBER NAME	DESCRIPTION
AssociateIndex	Specifies that the index for a specified topic is performed in the specified URL.
Find	Specifies that the search page of a specified URL is displayed.
Index	Specifies that the index of a specified URL is displayed.
KeywordIndex	Specifies a keyword to search for and the action to take in the specified URL.
TableOfContents	Specifies that the table of contents of the HTML 1.0 Help file is displayed.
Topic	Specifies that the topic referenced by the specified URL is displayed.

At run time, pressing F1 when the control—for which you have set the **HelpKeyword** and **HelpNavigator** properties—has focus will open the Help file you associated with that HelpProvider component.

Currently, the **HelpNamespace** property supports Help files in the following three formats: HTMLHelp 1.x, HTMLHelp 2.0, and HTML. Thus, you can set the **HelpNamespace** property to an `http://` address, such as a Web page. If this is done, it will open the default browser to the Web page with the string specified in the **HelpKeyword** property used as the anchor. The anchor is used to jump to a specific part of an HTML page.

IMPORTANT

Be careful to check any information that is sent from a client before using it in your application. Malicious users might try to send or inject executable script, SQL statements, or other code. Before you display a user's input, store it in a database, or work with it, check that it does not contain potentially unsafe information. A typical way to check is to use a regular expression to look for keywords such as "SCRIPT" when you receive input from a user.

You can also use the HelpProvider component to show pop-up Help, even if you have it configured to display Help files for the controls on your Windows Forms. For more information, see How to: Display Pop-up Help.

See also

- How to: Display Pop-up Help
- Control Help Using ToolTips
- Integrating User Help in Windows Forms
- Windows Forms

How to: Display pop-up Help

11/3/2020 • 2 minutes to read • Edit Online

One way to display Help on Windows Forms is through the **Help** button, located on the right side of the title bar, accessible through the HelpButton property. This type of Help display is well-suited for use with dialog boxes. Dialog boxes shown modally (with the ShowDialog method) have trouble bringing up external Help systems, because modal dialog boxes need to be closed before focus can shift to another window. Additionally, using the **Help** button requires that there is no **Minimize** button or **Maximize** button shown in the title bar. This is a standard dialog-box convention, whereas forms usually have **Minimize** and **Maximize** buttons.

You can also use the HelpProvider component to link controls to files in a Help system, even if you have implemented pop-up Help. For more information, see Providing Help in a Windows Application.

Display pop-up Help

1. In Visual Studio, drag a HelpProvider component from the Toolbox to your form.

   It will sit in the tray at the bottom of the Windows Forms Designer.

2. In the Properties window, set the HelpButton property to `true`. This will display a button with a question mark in it on the right side of the title bar of the form.

3. In order for the HelpButton to display, the form's MinimizeBox and MaximizeBox properties must be set to `false`, the ControlBox property set to `true`, and the FormBorderStyle property to one of the following values: FixedSingle, Fixed3D, FixedDialog or Sizable.

4. Select the control for which you want to show help on your form and set the Help string in the Properties window. This is the string of text that will be displayed in a window similar to a ToolTip.

5. Press **F5**.

6. Press the **Help** button on the title bar and click the control on which you set the Help string.

See also

- Control Help Using ToolTips
- Integrating User Help in Windows Forms
- Windows Forms

Control Help Using ToolTips

11/3/2020 • 2 minutes to read • Edit Online

You can use the ToolTip component to display a brief, specialized Help message for individual controls on Windows Forms. The ToolTip component provides a property that specifies the text displayed for each control on the form. For more information about working with the ToolTip component in this way, see How to: Set ToolTips for Controls on a Windows Form at Design Time. Additionally, you can configure the ToolTip component so that there is a delay before it is shown. For more information, see How to: Change the Delay of the Windows Forms ToolTip Component.

See also

- How to: Display Pop-up Help
- ToolTip Component
- Integrating User Help in Windows Forms
- Windows Forms

Windows Forms Accessibility

11/3/2020 • 2 minutes to read • Edit Online

The accessibility functionality of Windows Forms allows you to make your application available to a wide variety of users.

In This Section

Walkthrough: Creating an Accessible Windows-based Application
Describes all of the features you should support to increase accessibility.

Reference

Accessibility
A namespace containing a number of classes related to accessibility.

AccessibleObject
Provides information that accessibility applications use to adjust an application's user interface (UI) for users with impairments.

Related Sections

Providing Accessibility Information for Controls on a Windows Form
Describes how to supply information that Windows Forms controls can use to assist users with impairments.

Automatic Scaling in Windows Forms
Describes how to make your Windows Forms application react to changes in the system font size.

Walkthrough: Creating an Accessible Windows-based Application

3/9/2021 • 8 minutes to read • Edit Online

Creating an accessible application has important business implications. Many governments have accessibility regulations for software purchase. The Certified for Windows logo includes accessibility requirements. An estimated 30 million residents of the U.S. alone, many of them potential customers, are affected by the accessibility of software.

This walkthrough will address the five accessibility requirements for the Certified for Windows logo. According to these requirements, an accessible application will:

- Support Control Panel size, color, font, and input settings. The menu bar, title bar, borders, and status bar will all resize themselves when the user changes the control panel settings. No additional changes to the controls or code are required in this application.
- Support High Contrast mode.
- Provide documented keyboard access to all features.
- Expose location of the keyboard focus visually and programmatically.
- Avoid conveying important information by sound alone.

For more information, see Resources for Designing Accessible Applications.

For information on supporting varying keyboard layouts, see Best Practices for Developing World-Ready Applications.

Creating the Project

This walkthrough creates the user interface for an application that takes pizza orders. It consists of a TextBox for the customer's name, a RadioButton group to select the pizza size, a CheckedListBox for selecting the toppings, two Button controls labeled Order and Cancel, and a Menu with an Exit command.

The user enters the customer's name, the size of the pizza, and the toppings desired. When the user clicks the Order button, a summary of the order and its cost are displayed in a message box and the controls are cleared and ready for the next order. When the user clicks the Cancel button, the controls are cleared and ready for the next order. When the user clicks the Exit menu item, the program closes.

The emphasis of this walkthrough is not the code for a retail order system, but the accessibility of the user interface. The walkthrough demonstrates the accessibility features of several frequently used controls, including buttons, radio buttons, text boxes, and labels.

To begin making the application

- Create a new Windows Application in Visual Basic or Visual C#. Name the project **PizzaOrder**. For details, see Creating New Solutions and Projects.

Adding the Controls to the Form

When adding the controls to a form, keep in mind the following guidelines to make an accessible application:

- Set the AccessibleDescription and AccessibleName properties. In this example, the Default setting for the

AccessibleRole is sufficient. For more information on the accessibility properties, see Providing Accessibility Information for Controls on a Windows Form.

- Set the font size to 10 points or larger.

> **NOTE**
>
> If you set the font size of the form to 10 when you start, then all controls subsequently added to the form will have a font size of 10.

- Make sure any Label control that describes a TextBox control immediately precedes the TextBox control in the tab order.
- Add an access key, using the "&" character, to the Text property of any control the user may want to navigate to.
- Add an access key, using the "&" character, to the Text property of the label that precedes a control that the user may want to navigate to. Set the labels' UseMnemonic property to `true`, so that the focus is set to the next control in the tab order when the user presses the access key.
- Add access keys to all menu items.

To make your Windows Application accessible

- Add the controls to the form and set the properties as described below. See the picture at the end of the table for a model of how to arrange the controls on the form.

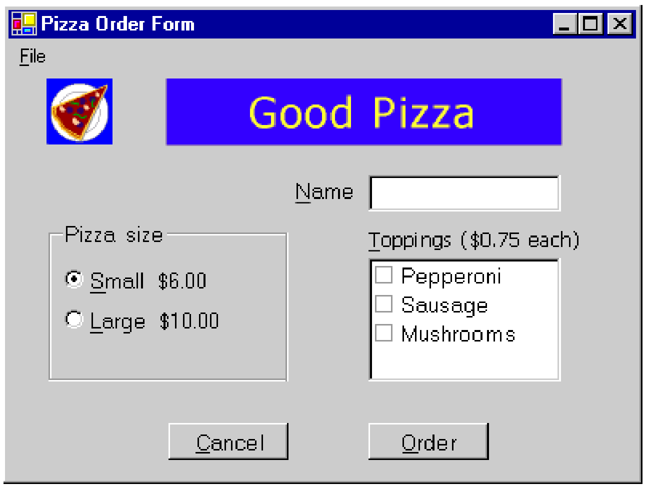
OBJECT	PROPERTY	VALUE
Form1	AccessibleDescription	Order form
	AccessibleName	Order form
	Font Size	10
	Text	Pizza Order Form
PictureBox	Name	logo
	AccessibleDescription	A slice of pizza
	AccessibleName	Company logo
	Image	Any icon or bitmap
Label	Name	companyLabel
	Text	Good Pizza
	TabIndex	1
	AccessibleDescription	Company name
	AccessibleName	Company name

OBJECT	PROPERTY	VALUE
	Backcolor	Blue
	Forecolor	Yellow
	Font size	18
Label	Name	customerLabel
	Text	&Name
	TabIndex	2
	AccessibleDescription	Customer name label
	AccessibleName	Customer name label
	UseMnemonic	True
TextBox	Name	customerName
	Text	(none)
	TabIndex	3
	AccessibleDescription	Customer name
	AccessibleName	Customer name
GroupBox	Name	sizeOptions
	AccessibleDescription	Pizza size options
	AccessibleName	Pizza size options
	Text	Pizza size
	TabIndex	4
RadioButton	Name	smallPizza
	Text	&Small $6.00
	Checked	True
	TabIndex	0
	AccessibleDescription	Small pizza
	AccessibleName	Small pizza

OBJECT	PROPERTY	VALUE
RadioButton	Name	largePizza
	Text	&Large $10.00
	TabIndex	1
	AccessibleDescription	Large pizza
	AccessibleName	Large pizza
Label	Name	toppingsLabel
	Text	&Toppings ($0.75 each)
	TabIndex	5
	AccessibleDescription	Toppings label
	AccessibleName	Toppings label
	UseMnemonic	True
CheckedListBox	Name	toppings
	TabIndex	6
	AccessibleDescription	Available toppings
	AccessibleName	Available toppings
	Items	Pepperoni, Sausage, Mushrooms
Button	Name	order
	Text	&Order
	TabIndex	7
	AccessibleDescription	Total the order
	AccessibleName	Total order
Button	Name	cancel
	Text	&Cancel
	TabIndex	8
	AccessibleDescription	Cancel the order

OBJECT	PROPERTY	VALUE
	AccessibleName	Cancel order
MainMenu	Name	theMainMenu
MenuItem	Name	fileCommands
	Text	&File
MenuItem	Name	exitApp
	Text	E&xit

Your form will look something like the following image:



Supporting High Contrast Mode

High Contrast mode is a Windows system setting that improves readability by using contrasting colors and font sizes that are beneficial for visually impaired users. The HighContrast property is provided to determine whether the High Contrast mode is set.

If SystemInformation.HighContrast is `true`, the application should:

- Display all user interface elements using the system color scheme
- Convey by visual cues or sound any information that is conveyed through color. For example, if particular list items are highlighted by using a red font, you could also add bold to the font, so that the user has a non-color cue that the items are highlighted.
- Omit any images or patterns behind text

The application should check the setting of HighContrast when the application starts and respond to the system event UserPreferenceChanged. The UserPreferenceChanged event is raised whenever the value of HighContrast changes.

In our application, the only element that is not using the system settings for color is `lblCompanyName`. The SystemColors class is used to change the color settings of the label to the user-selected system colors.

To enable High Contrast mode in an effective way

1. Create a method to set the colors of the label to the system colors.

```
Private Sub SetColorScheme()
    If SystemInformation.HighContrast Then
        companyLabel.BackColor = SystemColors.Window
        companyLabel.ForeColor = SystemColors.WindowText
    Else
        companyLabel.BackColor = Color.Blue
        companyLabel.ForeColor = Color.Yellow
    End If
End Sub
```

```
private void SetColorScheme()
{
    if (SystemInformation.HighContrast)
    {
        companyLabel.BackColor = SystemColors.Window;
        companyLabel.ForeColor = SystemColors.WindowText;
    }
    else
    {
        companyLabel.BackColor = Color.Blue;
        companyLabel.ForeColor = Color.Yellow;
    }
}
```

2. Call the `SetColorScheme` procedure in the form constructor (`Public Sub New()` in Visual Basic and `public Form1()` in Visual C#). To access the constructor in Visual Basic, you will need to expand the region labeled **Windows Form Designer generated code**.

```
Public Sub New()
    MyBase.New()
    InitializeComponent()
    SetColorScheme()
End Sub
```

```
public Form1()
{
    InitializeComponent();
    SetColorScheme();
}
```

3. Create an event procedure, with the appropriate signature, to respond to the UserPreferenceChanged event.

```
Protected Sub UserPreferenceChanged(sender As Object, _
e As Microsoft.Win32.UserPreferenceChangedEventArgs)
    SetColorScheme()
End Sub
```

```
public void UserPreferenceChanged(object sender,
Microsoft.Win32.UserPreferenceChangedEventArgs e)
{
    SetColorScheme();
}
```

4. Add code to the form constructor, after the call to `InitializeComponents`, to hook up the event procedure to the system event. This method calls the `SetColorScheme` procedure.

```
Public Sub New()
    MyBase.New()
    InitializeComponent()
    SetColorScheme()
    AddHandler Microsoft.Win32.SystemEvents.UserPreferenceChanged, _
       AddressOf Me.UserPreferenceChanged
End Sub
```

```
public Form1()
{
    InitializeComponent();
    SetColorScheme();
    Microsoft.Win32.SystemEvents.UserPreferenceChanged
       += new Microsoft.Win32.UserPreferenceChangedEventHandler(
       this.UserPreferenceChanged);
}
```

5. Add code to the form Dispose method, before the call to the Dispose method of the base class, to release the event when the application closes. To access the Dispose method in Visual Basic, you will need to expand the region labeled Windows Form Designer generated code.

> **NOTE**
>
> The system event code runs a thread separate from the main application. If you do not release the event, the code that you hook up to the event will run even after the program is closed.

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        components.Dispose()
    End If
    RemoveHandler Microsoft.Win32.SystemEvents.UserPreferenceChanged, _
       AddressOf Me.UserPreferenceChanged
    MyBase.Dispose(disposing)
End Sub
```

```
protected override void Dispose(bool disposing)
{
    if(disposing && components != null)
    {
        components.Dispose();
    }
    Microsoft.Win32.SystemEvents.UserPreferenceChanged
       -= new Microsoft.Win32.UserPreferenceChangedEventHandler(
       this.UserPreferenceChanged);
    base.Dispose( disposing );
}
```

6. Press F5 to run the application.

Conveying Important Information by Means Other Than Sound

In this application, no information is conveyed by sound alone. If you use sound in your application, then you should supply the information by some other means as well.

To supply information by some other means than sound

1. Make the title bar flash by using the Windows API function FlashWindow. For an example of how to call

Windows API functions, see Walkthrough: Calling Windows APIs.

> **NOTE**
>
> The user may have the Windows SoundSentry service enabled, which will also cause the window to flash when the system sounds are played through the computer's built-in speaker.

2. Display the important information in a non-modal window so that the user may respond to it.

3. Display a message box that acquires the keyboard focus. Avoid this method when the user may be typing.

4. Display a status indicator in the status notification area of the taskbar. For details, see Adding Application Icons to the TaskBar with the Windows Forms NotifyIcon Component.

Testing the Application

Before deploying the application, you should test the accessibility features that you have implemented.

To test accessibility features

1. To test keyboard access, unplug the mouse and navigate the user interface for each feature using only the keyboard. Ensure that all tasks may be performed using the keyboard only.

2. To test support of High Contrast, choose the Accessibility Options icon in Control Panel. Click the Display tab and select the Use High Contrast check box. Navigate through all user interface elements to ensure that the color and font changes are reflected. Also, ensure that images or patterns drawn behind text are omitted.

   > **NOTE**
   >
   > Windows NT 4 does not have an Accessibility Options icon in Control Panel. Thus, this procedure for changing the SystemInformation.HighContrast setting does not work in Windows NT 4.

3. Other tools are readily available for testing the accessibility of an application.

4. To test exposing the keyboard focus, run Magnifier. (To open it, click the **Start** menu, point to **Programs**, point to **Accessories**, point to **Accessibility**, and then click **Magnifier**). Navigate the user interface using both keyboard tabbing and the mouse. Ensure that all navigation is tracked properly in **Magnifier**.

5. To test exposing screen elements, run Inspect, and use both the mouse and the TAB key to reach each element. Ensure that the information presented in the Name, State, Role, Location, and Value fields of the Inspect window is meaningful to the user for each object in the UI.

Properties on Windows Forms Controls That Support Accessibility Guidelines

11/3/2020 • 2 minutes to read • Edit Online

Controls on the standard toolbox for Windows Forms support many of the accessibility guidelines, including exposing the keyboard focus and exposing the screen elements.

Planning Ahead for Accessibility

The controls' properties can be used to support other accessibility guidelines as shown in the following table. Additionally, you should use menus to provide access to program features.

CONTROL PROPERTY	CONSIDERATIONS FOR ACCESSIBILITY
AccessibleDescription	The description is reported to accessibility aids such as screen readers. Accessibility aids are specialized programs and devices that help people with disabilities use computers more effectively.
AccessibleName	The name that will be reported to the accessibility aids.
AccessibleRole	Describes the use of the element in the user interface.
TabIndex	Creates a sensible navigational path through the form. It is important for controls without intrinsic labels, such as text boxes, to have their associated label immediately precede them in the tab order.
Text	Use the "&" character to create access keys. Using access keys is part of providing documented keyboard access to features.
Font Size	If the font size is not adjustable, then it should be set to 10 points or larger. Once the form's font size is set, all the controls added to the form thereafter will have the same size.
Forecolor	If this property is set to the default, then the user's color preferences will be used on the form.
Backcolor	If this property is set to the default, then the user's color preferences will be used on the form.
BackgroundImage	Leave this property blank to make text more readable.

See also

- Walkthrough: Creating an Accessible Windows-based Application

Use WPF controls in Windows Forms apps

3/9/2021 • 2 minutes to read • Edit Online

You can use Windows Presentation Foundation (WPF) controls in Windows Forms-based applications. Although these are two different view technologies, they interoperate smoothly.

The Windows Forms Designer in Visual Studio provides a visual design environment for hosting Windows Presentation Foundation controls. A WPF control is hosted by a special Windows Forms control that's named ElementHost. This control enables the WPF control to participate in the form's layout and to receive keyboard and mouse messages. At design time, you can arrange the ElementHost control just as you would any Windows Forms control.

You can also use Windows Forms controls in WPF-based applications. For more information, see Design XAML in Visual Studio.

See also

- WPF and Windows Forms interoperation

How to: Copy and paste an ElementHost control

3/9/2021 • 2 minutes to read • Edit Online

This procedure shows you how to copy a Windows Presentation Foundation (WPF) control on a Windows Form in Visual Studio.

1. In Visual Studio, add a new WPF UserControl to a Windows Forms project. Use the default name for the control type, `UserControl1.xaml`. For more information, see Walkthrough: Creating New WPF Content on Windows Forms at Design Time.
2. In the **Properties** window, set the value of the Width and Height properties of `UserControl1` to **200**.
3. Set the value of the Background property to **Blue**.
4. Build the project.
5. Open `Form1` in the Windows Forms Designer.
6. From the **Toolbox**, drag an instance of `UserControl1` onto the form.

   An instance of `UserControl1` is hosted in a new ElementHost control named `elementHost1`.
7. With `elementHost1` selected, press **Ctrl**+**C** to copy it to the clipboard.
8. Press **Ctrl**+**V** to paste the copied control onto the form.

   A new ElementHost control named `elementHost2` is created on the form.

See also

- ElementHost
- WindowsFormsHost
- Migration and Interoperability
- Using WPF Controls
- Design XAML in Visual Studio

Walkthrough: Arrange WPF content on Windows Forms at design time

3/9/2021 • 3 minutes to read • Edit Online

This article shows you how to use the Windows Forms layout features, such as anchoring and snaplines, to arrange Windows Presentation Foundation (WPF) controls.

Prerequisites

You need Visual Studio to complete this walkthrough.

Create the project

Open Visual Studio and create a new Windows Forms Application project in Visual Basic or Visual C# named `ArrangeElementHost`.

> **NOTE**
> When hosting WPF content, only C# and Visual Basic projects are supported.

Create the WPF control

After you add a WPF control to the project, you can arrange it on the form.

1. Add a new WPF UserControl to the project. Use the default name for the control type, `UserControl1.xaml`. For more information, see Walkthrough: Creating New WPF Content on Windows Forms at Design Time.

2. In Design view, make sure that `UserControl1` is selected.

3. In the **Properties** window, set the value of the Width and Height properties to **200**.

4. Set the value of the Background property to **Blue**.

5. Build the project.

Host WPF controls in a layout panel

You can use WPF controls in layout panels in the same way you use other Windows Forms controls.

1. Open `Form1` in the Windows Forms Designer.

2. In the **Toolbox**, drag a TableLayoutPanel control onto the form.

3. On the TableLayoutPanel control's smart tag panel, select **Remove Last Row**.

4. Resize the TableLayoutPanel control to a larger width and height.

5. In the **Toolbox**, double-click `UserControl1` to create an instance of `UserControl1` in the first cell of the TableLayoutPanel control.

   The instance of `UserControl1` is hosted in a new ElementHost control named `elementHost1`.

6. In the **Toolbox**, double-click `UserControl1` to create another instance in the second cell of the

TableLayoutPanel control.

7. In the **Document Outline** window, select `tableLayoutPanel1`.

8. In the **Properties** window, set the value of the Padding property to **10, 10, 10, 10**.

   Both ElementHost controls are resized to fit into the new layout.

Use snaplines to align WPF controls

Snaplines enable easy alignment of controls on a form. You can use snaplines to align your WPF controls as well. For more information, see Walkthrough: Arranging Controls on Windows Forms Using Snaplines.

1. From the **Toolbox**, drag an instance of `UserControl1` onto the form, and place it in the space beneath the TableLayoutPanel control.

   The instance of `UserControl1` is hosted in a new ElementHost control named `elementHost3`.

2. Using snaplines, align the left edge of `elementHost3` with the left edge of TableLayoutPanel control.

3. Using snaplines, size `elementHost3` to the same width as the TableLayoutPanel control.

4. Move `elementHost3` toward the TableLayoutPanel control until a center snapline appears between the controls.

5. In the **Properties** window, set the value of the Margin property to **20, 20, 20, 20**.

6. Move the `elementHost3` away from the TableLayoutPanel control until the center snapline appears again. The center snapline now indicates a margin of 20.

7. Move `elementHost3` to the right until its left edge aligns with the left edge of `elementHost1`.

8. Change the width of `elementHost3` until its right edge aligns with the right edge of `elementHost2`.

Anchor and dock WPF controls

A WPF control hosted on a form has the same anchoring and docking behavior as other Windows Forms controls.

1. Select `elementHost1`.

2. In the **Properties** window, set the Anchor property to **Top, Bottom, Left, Right**.

3. Resize the TableLayoutPanel control to a larger size.

   The `elementHost1` control resizes to fill the cell.

4. Select `elementHost2`.

5. In the **Properties** window, set the value of the Dock property to Fill.

   The `elementHost2` control resizes to fill the cell.

6. Select the TableLayoutPanel control.

7. Set the value of its Dock property to Top.

8. Select `elementHost3`.

9. Set the value of its Dock property to Fill.

   The `elementHost3` control resizes to fill the remaining space on the form.

10. Resize the form.

    All three ElementHost controls resize appropriately.

    For more information, see How to: Anchor and Dock Child Controls in a TableLayoutPanel Control.

See also

- ElementHost
- WindowsFormsHost
- How to: Anchor and Dock Child Controls in a TableLayoutPanel Control
- How to: Align a Control to the Edges of Forms at Design Time
- Walkthrough: Arranging Controls on Windows Forms Using Snaplines
- Migration and Interoperability
- Using WPF Controls
- Design XAML in Visual Studio

Walkthrough: Create new WPF content on Windows Forms at design time

3/9/2021 • 2 minutes to read • Edit Online

This article shows you how to create a Windows Presentation Foundation (WPF) control for use in your Windows Forms-based applications.

Prerequisites

You need Visual Studio to complete this walkthrough.

Create the project

Open Visual Studio and create a new **Windows Forms App (.NET Framework)** project in Visual Basic or Visual C# named `HostingWpf`.

> **NOTE**
>
> When hosting WPF content, only C# and Visual Basic projects are supported.

Create a new WPF control

Creating a new WPF control and adding it to your project is as easy as adding any other item to your project. The Windows Forms Designer works with a particular kind of control named *composite control*, or *user control*. For more information about WPF user controls, see UserControl.

> **NOTE**
>
> The System.Windows.Controls.UserControl type for WPF is distinct from the user control type provided by Windows Forms, which is also named System.Windows.Forms.UserControl.

To create a new WPF control:

1. In **Solution Explorer**, add a new **WPF User Control Library (.NET Framework)** project to the solution. Use the default name for the control library, `WpfControlLibrary1`. The default control name is `UserControl1.xaml`.

   Adding the new control has the following effects:

   - File UserControl1.xaml is added.
   - File UserControl1.xaml.cs (or UserControl1.xaml.vb) is added. This file contains the code-behind for event handlers and other implementation.
   - References to WPF assemblies are added.
   - File UserControl1.xaml opens in the WPF Designer for Visual Studio.

2. In Design view, make sure that `UserControl1` is selected.
3. In the **Properties** window, set the value of the Width and Height properties to **200**.

4. From the **Toolbox**, drag a System.Windows.Controls.TextBox control onto the design surface.

5. In the **Properties** window, set the value of the Text property to **Hosted Content**.

> **NOTE**
>
> In general, you should host more sophisticated WPF content. The System.Windows.Controls.TextBox control is used here for illustrative purposes only.

6. Build the project.

Add a WPF control to a Windows Form

Your new WPF control is ready for use on the form. Windows Forms uses the ElementHost control to host WPF content.

To add a WPF control to a Windows Form:

1. Open `Form1` in the Windows Forms Designer.

2. In the **Toolbox**, find the tab labeled **WPFUserControlLibrary WPF User Controls**.

3. Drag an instance of `UserControl1` onto the form.

   - An ElementHost control is created automatically on the form to host the WPF control.

   - The ElementHost control is named `elementHost1` and in the **Properties** window, you can see its Child property is set to **UserControl1**.

   - References to WPF assemblies are added to the project.

   - The `elementHost1` control has a smart tag panel that shows the available hosting options.

4. In the **ElementHost Tasks** smart tag panel, select **Dock in parent container**.

5. Press **F5** to build and run the application.

Next steps

Windows Forms and WPF are different technologies, but they are designed to interoperate closely. To provide richer appearance and behavior in your applications, try the following:

- Host a Windows Forms control in a WPF page. For more information, see Walkthrough: Hosting a Windows Forms Control in WPF.

- Apply Windows Forms visual styles to your WPF content. For more information, see How to: Enable Visual Styles in a Hybrid Application.

- Change the style of your WPF content. For more information, see Walkthrough: Styling WPF Content.

See also

- ElementHost
- WindowsFormsHost
- Migration and Interoperability
- Using WPF Controls
- Design XAML in Visual Studio

Walkthrough: Assign WPF content on Windows Forms at design time

3/9/2021 • 2 minutes to read • Edit Online

This article show you how to select the Windows Presentation Foundation (WPF) control types you want to display on your form. You can select any WPF control types that are included in your project.

Prerequisites

You need Visual Studio to complete this walkthrough.

Create the project

Open Visual Studio and create a new Windows Forms Application project in Visual Basic or Visual C# named `SelectingWpfContent`.

> **NOTE**
> When hosting WPF content, only C# and Visual Basic projects are supported.

Create the WPF control types

After you add WPF control types to the project, you can host them in different ElementHost controls.

1. Add a new WPF UserControl project to the solution. Use the default name for the control type, `UserControl1.xaml`. For more information, see Walkthrough: Creating New WPF Content on Windows Forms at Design Time.
2. In Design view, make sure that `UserControl1` is selected.
3. In the **Properties** window, set the value of the Width and Height properties to **200**.
4. Add a System.Windows.Controls.TextBox control to the UserControl and set the value of the Text property to **Hosted Content**.
5. Add a second WPF UserControl to the project. Use the default name for the control type, `UserControl2.xaml`.
6. In the **Properties** window, set the value of the Width and Height properties to **200**.
7. Add a System.Windows.Controls.TextBox control to the UserControl and set the value of the Text property to **Hosted Content 2**.

   > **NOTE**
   > In general, you should host more sophisticated WPF content. The System.Windows.Controls.TextBox control is used here for illustrative purposes only.

8. Build the project.

Select WPF controls

You can assign different WPF content to an ElementHost control, which is already hosting content.

1. Open `Form1` in the Windows Forms Designer.
2. In the **Toolbox**, double-click `UserControl1` to create an instance of `UserControl1` on the form.

   An instance of `UserControl1` is hosted in a new ElementHost control named `elementHost1`.
3. In the smart tag panel for `elementHost1`, open the **Select Hosted Content** drop-down list.
4. Select **UserControl2** from the drop-down list box.

   The `elementHost1` control now hosts an instance of the `UserControl2` type.
5. In the **Properties** window, confirm that the Child property is set to **UserControl2**.
6. From the **Toolbox**, in the **WPF Interoperability** group, drag an ElementHost control onto the form.

   The default name for the new control is `elementHost2`.
7. In the smart tag panel for `elementHost2`, open the **Select Hosted Content** drop-down list.
8. Select **UserControl1** from the drop-down list.
9. The `elementHost2` control now hosts an instance of the `UserControl1` type.

See also

- ElementHost
- WindowsFormsHost
- Migration and Interoperability
- Using WPF Controls
- Design XAML in Visual Studio

Walkthrough: Style WPF content

3/9/2021 • 2 minutes to read • Edit Online

This article show you how to apply styling to a Windows Presentation Foundation (WPF) control hosted on a Windows Form.

Prerequisites

You need Visual Studio to complete this walkthrough.

Create the project

Open Visual Studio and create a new Windows Forms Application project in Visual Basic or Visual C# named `StylingWpfContent`.

> **NOTE**
> When hosting WPF content, only C# and Visual Basic projects are supported.

Create the WPF control types

After you add a WPF control type to the project, you can host it in an ElementHost control.

1. Add a new WPF UserControl project to the solution. Use the default name for the control type, `UserControl1.xaml`. For more information, see Walkthrough: Creating New WPF Content on Windows Forms at Design Time.

2. In Design view, make sure that `UserControl1` is selected.

3. In the **Properties** window, set the value of the Width and Height properties to **200**.

4. Add a System.Windows.Controls.Button control to the UserControl and set the value of the Content property to **Cancel**.

5. Add a second System.Windows.Controls.Button control to the UserControl and set the value of the Content property to **OK**.

6. Build the project.

Apply a Style to a WPF Control

You can apply different styling to a WPF control to change its appearance and behavior.

1. Open `Form1` in the Windows Forms Designer.

2. In the **Toolbox**, double-click `UserControl1` to create an instance of `UserControl1` on the form.

   An instance of `UserControl1` is hosted in a new ElementHost control named `elementHost1`.

3. In the smart tag panel for `elementHost1`, click **Edit Hosted Content** from the drop-down list.

   `UserControl1` opens in the WPF Designer.

4. In XAML view, insert the following XAML after the `<UserControl>` opening tag. This XAML creates a

gradient with a contrasting gradient border. When the control is clicked, the gradients are changed to generate a pressed button look. For more information, see Styling and Templating.

```
<UserControl.Resources>
 <LinearGradientBrush x:Key="NormalBrush" EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#FFF" Offset="0.0"/>
     <GradientStop Color="#CCC" Offset="1.0"/>
 </LinearGradientBrush>
 <LinearGradientBrush x:Key="PressedBrush" EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#BBB" Offset="0.0"/>
     <GradientStop Color="#EEE" Offset="0.1"/>
     <GradientStop Color="#EEE" Offset="0.9"/>
     <GradientStop Color="#FFF" Offset="1.0"/>
 </LinearGradientBrush>
 <LinearGradientBrush x:Key="NormalBorderBrush" EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#CCC" Offset="0.0"/>
     <GradientStop Color="#444" Offset="1.0"/>
 </LinearGradientBrush>
 <LinearGradientBrush x:Key="BorderBrush" EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#CCC" Offset="0.0"/>
     <GradientStop Color="#444" Offset="1.0"/>
 </LinearGradientBrush>
 <LinearGradientBrush x:Key="PressedBorderBrush" EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#444" Offset="0.0"/>
     <GradientStop Color="#888" Offset="1.0"/>
 </LinearGradientBrush>

 <Style x:Key="SimpleButton" TargetType="{x:Type Button}" BasedOn="{x:Null}">
     <Setter Property="Background" Value="{StaticResource NormalBrush}"/>
     <Setter Property="BorderBrush" Value="{StaticResource NormalBorderBrush}"/>
     <Setter Property="Template">
         <Setter.Value>
             <ControlTemplate TargetType="{x:Type Button}">
                 <Grid x:Name="Grid">
                     <Border x:Name="Border" Background="{TemplateBinding Background}" BorderBrush="
{TemplateBinding BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}" Padding="
{TemplateBinding Padding}"/>
                     <ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}" VerticalAlignment="{TemplateBinding
VerticalContentAlignment}" RecognizesAccessKey="True"/>
                 </Grid>
                 <ControlTemplate.Triggers>
                     <Trigger Property="IsPressed" Value="true">
                         <Setter Property="Background" Value="{StaticResource PressedBrush}"
TargetName="Border"/>
                         <Setter Property="BorderBrush" Value="{StaticResource PressedBorderBrush}"
TargetName="Border"/>
                     </Trigger>
                 </ControlTemplate.Triggers>
             </ControlTemplate>
         </Setter.Value>
     </Setter>
 </Style>
</UserControl.Resources>
```

5. Apply the `SimpleButton` style defined in the previous step to the Cancel button by inserting the following XAML in the `<Button>` tag of the **Cancel** button.

```
Style="{StaticResource SimpleButton}
```

Your button declaration will resemble the following XAML:

```
<Button Height="23" Margin="41,52,98,0" Name="button1" VerticalAlignment="Top"
             Style="{StaticResource SimpleButton}">Cancel</Button>
```

6. Build the project.
7. Open `Form1` in the Windows Forms Designer.
8. The new style is applied to the button control.
9. From the **Debug** menu, select **Start Debugging** to run the application.
10. Click the **OK** and **Cancel** buttons and view the differences.

See also

- ElementHost
- WindowsFormsHost
- Migration and Interoperability
- Using WPF Controls
- Design XAML in Visual Studio
- XAML Overview (WPF)
- Styling and Templating