Ordering Information: **Visual Basic .NET How to Program, Second Edition**

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ONLINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html** To learn more about Deitel instructor-led corporate training delivered at your location, visit **www.deitel.com/training** or contact Christi Kelsey at (978) 461-5880 or e-mail: **christi.kelsey@deitel.net**.

Note from the Authors: This article is an excerpt from Chapter 16, Section 16.3 of *Visual Basic .NET How to Program, 2/e*. This article discuss the Framework Class Library (FCL) and how it provides the color structure in Visual Basic .NET to create and manage colors. Readers of this article should be familiar with Visual Basic .NET programming. The code examples included in this article show readers examples using the Deitel signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

16.3 Color Control

Colors can enhance a program's appearance and help convey meaning. For example, a red traffic light indicates stop, yellow indicates caution and green indicates go. The Framework Class Library (FCL) provides the ***Color*** structure for creating and manipulating colors. Colors are created by combining alpha, red, green and blue components. Together, these components are called *ARGB values*. Each component in an ARGB value is an integer in the range 0 to 255 inclusive. The alpha value determines the color's opacity. For example, the alpha value 0 results in a transparent color, whereas the value 255 results in an opaque (i.e., solid) color. Alpha values between 1 and 254, inclusive, result in a blending effect of the color's Red-Green-Blue (RGB) value with that of any background color, causing a semi-transparent effect. The first number in the RGB value defines the amount of red in the color, the second defines the amount of green and the third defines the amount of blue. The larger the value, the greater the amount of that particular color. Visual Basic .NET enables programmers to choose from approximately 17 million colors. Figure 16.3 summarizes some predefined **Color** constants, and Fig. 16.4 describes several **Color** methods and properties.

The table in Fig. 16.4 describes two ***FromArgb*** method calls. One takes three **Integer** arguments, and one takes four **Integer** arguments (all argument values must be between 0 and 255, inclusive). Both take **Integer** arguments specifying the amount of red, green and blue. The overloaded version takes four arguments and allows the user to specify the alpha value; the three-argument version defaults the alpha to 255 (opaque). Both methods return a **Color** value that corresponds to the specified values. **Color** properties ***A***, ***R***, ***G*** and ***B*** return **Byte**s that represent the values from 0 to 255, corresponding to the amounts of alpha, red, green and blue, respectively.

Programmers draw shapes and **String**s using **Brush**es and **Pen**s. A ***Pen***, which functions similarly to an ordinary pen, is used to draw lines. Most drawing methods require a **Pen** object. The overloaded **Pen** constructors allow programmers to specify the colors and widths of the lines that they wish to draw. The **System.Drawing** namespace also provides a ***Pens*** collection containing predefined **Pen**s.

All classes derived from abstract class ***Brush*** define classes that color the interiors of graphical shapes (for example, the **SolidBrush** constructor takes a **Color** value that represents the drawing color). In most fill methods (e.g., ***FillRectangle***, which draws a solid rectangle), **Brush**es fill a space with a color, pattern or image. Figure 16.5 summarizes various **Brush**es and their functions.

Constants in structure **Color** (all are **Public Shared**)	RGB value	Constants in structure **Color** (all are **Public Shared**)	RGB value
`Orange`	255, 200, 0	`White`	255, 255, 255
`Pink`	255, 175, 175	`Gray`	28, 128, 128
`Cyan`	0, 255, 255	`DarkGray`	64, 64, 64

Fig. 16.3 Some **Color** structure **Shared** constants and their RGB values. (Part 1 of 2)

Constants in structure `Color` (all are `Public Shared`)	RGB value	Constants in structure `Color` (all are `Public Shared`)	RGB value
`Magenta`	255, 0, 255	`Red`	255, 0, 0
`Yellow`	255, 255, 0	`Green`	0, 255, 0
`Black`	0, 0, 0	`Blue`	0, 0, 255

Fig. 16.3 Some `Color` structure `Shared` constants and their RGB values. (Part 2 of 2)

Structure `Color` methods and properties	Description
Common Methods	
`Shared FromArgb`	Creates a color based on red, green and blue values expressed as `Integers` from 0 to 255. Overloaded version allows specification of alpha, red, green and blue values.
`Shared FromName`	Creates a color from a name, passed as a `String`.
Common Properties	
`A`	`Integer` between 0 and 255, representing the alpha component.
`R`	`Integer` between 0 and 255, representing the red component.
`G`	`Integer` between 0 and 255, representing the green component.
`B`	`Integer` between 0 and 255, representing the blue component.

Fig. 16.4 `Color` structure members.

The application in Fig. 16.6 demonstrates several of the methods described in Fig. 16.4. It displays two overlapping rectangles, allowing the user to experiment with color values and color names.

Class	Description
`HatchBrush`	Uses a rectangular brush to fill a region with a pattern. The pattern is defined by a member of the `HatchStyle` enumeration, a foreground color (with which the pattern is drawn) and a background color.
`LinearGradientBrush`	Fills a region with a gradual blend of one color into another. Linear gradients are defined along a line. They can be specified by the two colors, the angle of the gradient and either the width of a rectangle or two points.

Fig. 16.5 Some classes that derive from class `Brush`. (Part 1 of 2)

Class	Description
SolidBrush	Fills a region with one color. Defined by a **Color** value.
TextureBrush	Fills a region by repeating a specified **Image** across the surface.

Fig. 16.5 Some classes that derive from class **Brush**. (Part 2 of 2)

```
1  ' Fig. 16.6: ShowColors.vb
2  ' Using different colors in Visual Basic.
3
4  Public Class FrmColorForm
5     Inherits System.Windows.Forms.Form
6
7     ' input text boxes
8     Friend WithEvents txtColorName As TextBox
9     Friend WithEvents txtGreenBox As TextBox
10    Friend WithEvents txtRedBox As TextBox
11    Friend WithEvents txtAlphaBox As TextBox
12    Friend WithEvents txtBlueBox As TextBox
13
14    ' set color command buttons
15    Friend WithEvents cmdColorName As Button
16    Friend WithEvents cmdColorValue As Button
17
18    ' color labels
19    Friend WithEvents lblBlue As Label
20    Friend WithEvents lblGreen As Label
21    Friend WithEvents lblRed As Label
22    Friend WithEvents lblAlpha As Label
23
24    ' group boxes
25    Friend WithEvents nameBox As GroupBox
26    Friend WithEvents colorValueGroup As GroupBox
27
28    ' Visual Studio .NET generated code
29
30    ' color for back rectangle
31    Private mBehindColor As Color = Color.Wheat
32
33    ' color for front rectangle
34    Private mFrontColor As Color = Color.FromArgb(100, 0, 0, 255)
35
36    ' overrides Form OnPaint method
37    Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
38       Dim graphicsObject As Graphics = e.Graphics ' get graphics
39
```

Fig. 16.6 Color value and alpha demonstration. (Part 1 of 3)

```
40          Dim textBrush As SolidBrush = _
41             New SolidBrush(Color.Black) ' create text brush
42
43          Dim brush As SolidBrush = _
44             New SolidBrush(Color.White) ' create solid brush
45
46          ' draw white background
47          graphicsObject.FillRectangle(brush, 4, 4, 275, 180)
48
49          ' display name of behindColor
50          graphicsObject.DrawString(mBehindColor.Name, Me.Font, _
51             textBrush, 40, 5)
52
53          ' set brush color and display back rectangle
54          brush.Color = mBehindColor
55
56          graphicsObject.FillRectangle(brush, 45, 20, 150, 120)
57
58          ' display Argb values of front color
59          graphicsObject.DrawString("Alpha: " & mFrontColor.A & _
60             " Red: " & mFrontColor.R & " Green: " & mFrontColor.G _
61             & " Blue: " & mFrontColor.B, Me.Font, textBrush, _
62             55, 165)
63
64          ' set brush color and display front rectangle
65          brush.Color = mFrontColor
66
67          graphicsObject.FillRectangle(brush, 65, 35, 170, 130)
68       End Sub ' OnPaint
69
70       ' handle cmdColorValue click event
71       Private Sub cmdColorValue_Click(ByVal sender As _
72          System.Object, ByVal e As System.EventArgs) _
73          Handles cmdColorValue.Click
74
75          ' obtain new front color from text boxes
76          mFrontColor = Color.FromArgb(txtAlphaBox.Text, _
77             txtRedBox.Text, txtGreenBox.Text, txtBlueBox.Text)
78
79          Invalidate() ' refresh Form
80       End Sub ' cmdColorValue_Click
81
82       Private Sub cmdColorName_Click(ByVal sender As _
83          System.Object, ByVal e As System.EventArgs) _
84          Handles cmdColorName.Click
85
86          ' set behindColor to color specified in text box
87          mBehindColor = Color.FromName(txtColorName.Text)
88
89          Invalidate() ' refresh Form
90       End Sub ' cmdColorName_Click
91
92    End Class ' FrmColorForm
```

Fig. 16.6 Color value and alpha demonstration. (Part 2 of 3)

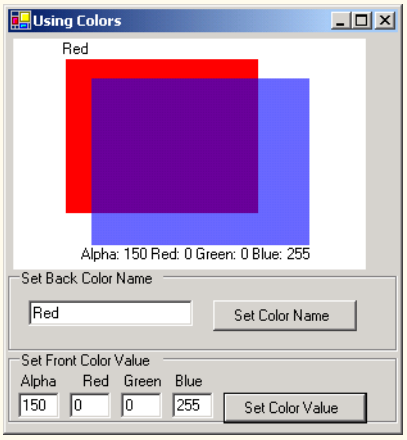

Fig. 16.6 Color value and alpha demonstration. (Part 3 of 3)

When the application begins its execution, it calls class **`FrmColorForm`**' **`OnPaint`** method to draw (i.e., paint) the form. A Visual Basic *graphics context* represents a drawing surface and enables drawing on the form. A ***`Graphics`*** object manages a graphics context by controlling how information is drawn. Line 38 retrieves a reference to **`PaintEventArgs e`**'s **`Graphics`** object and assigns it to **`Graphics`** reference **`graphicsObject`**. Lines 40–44 create a black and a white **`SolidBrush`** for drawing solid shapes on the form.

`Graphics` method **`FillRectangle`** draws a solid white rectangle with the **`Brush`** supplied as a parameter (line 47). It takes as parameters a brush, the *x*- and *y*-coordinates of a point and the width and height of the rectangle to draw. The point represents the upper-left corner of the rectangle. Lines 50–51 display the background color's **`Name`** by calling **`Graphics`** method **`DrawString`**. The FCL provides several overloaded **`DrawString`** methods; the version demonstrated in lines 50–51 takes a **`String`** to display, the display **`Font`**, a **`Brush`** and the *x*- and *y*-coordinates corresponding to where the **`String`**'s first character will be displayed.

Lines 54 and 56 assign **`mBehindColor`**'s value to the **`Brush`**'s **`Color`** property and draw a solid rectangle using that brush. Lines 59–67 retrieve and display the ARGB values of **`Color`** **`mFrontColor`** and then draw a filled rectangle that overlaps the first rectangle.

`Button` event-handler method **`cmdColorValue_Click`** (lines 71–80) calls **`Color`** method **`FromArgb`** to create a new **`Color`** value from the ARGB values specified by users. The newly created **`Color`** is then assigned to **`mFrontColor`**. **`Button`** event-handler method **`cmdColorName_Click`** (lines 82–90) calls **`Color`** method **`FromName`** to create a new **`Color`** value from the **`colorName`** specified in the **`txtColorName`** text box. This **`Color`** is assigned to **`mBehindColor`**. Method **`OnPaint`** is invoked indirectly by calling method ***`Invalidate`*** (inherited from **`Control`**) in lines 79 and 89.

If users assign alpha value between 0 and 255 for **`mFrontColor`**, the effects of alpha blending are apparent. In the screenshot, the red rectangle blends with the blue rectangle to create purple where the two overlap.