Everything you wanted to know about Visual Basic 6 Colors

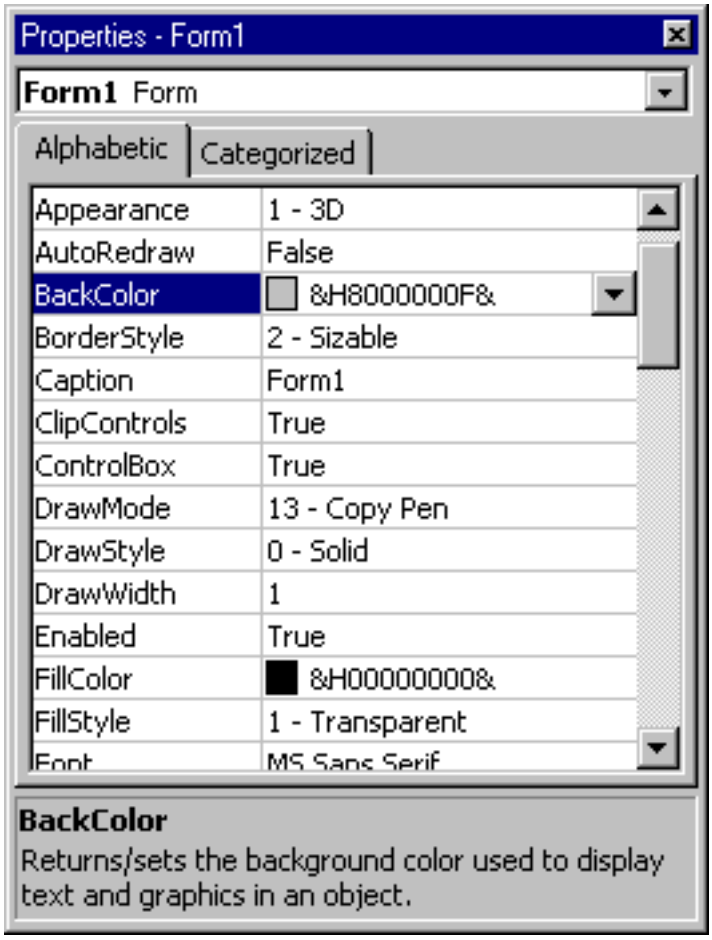
The topic of beautifying your Visual Basic program is always a popular one in my classes. If you are like most of my Visual Basic students, right about now you are clamoring to make your programs prettier. Personally, I'm pretty much a believer in accepting the default colors that Visual Basic assigns. As you will see during the course of the book, these default colors are the colors the user has selected in the Control Panel of their PC. But invariably, beginners love to experiment with features such as colors and Font styles. So these are the topics I'll cover in this handout.

Colors

During the course of the book you'll see that many of the objects in Visual Basic have color properties---most notably, the BackColor and ForeColor properties. Again, if you accept the default values for these properties, your object will exhibit the same colors that the user has selected in their Windows Control Panel. Most importantly, if the user then changes their color selections, the colors in your program will change accordingly.

Is there a default color?

Yes, each object in Visual Basic that has a color property already has a default value established for it when you create the control. Let’s take a look at these default values now. If you want, while you're reading about the default colors, place some controls on a from. How about a Command Button, label, text box, list box, CheckBox, and an option button? Now bring up the Properties Window and examine the value for the **BackColor** property of the form. What do you see?



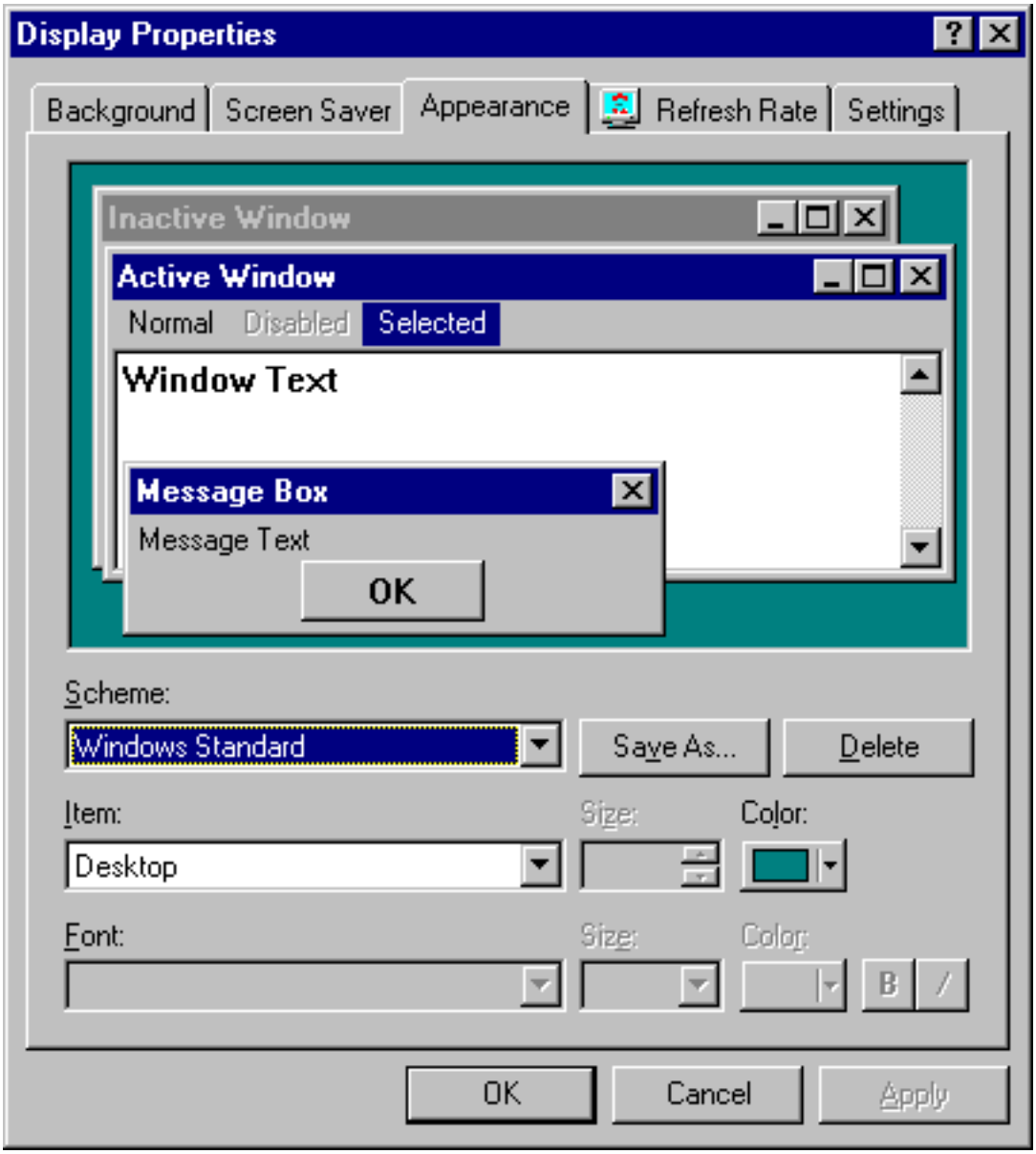
Here's the number that I see in the Properties Window.

&H8000000F&

If you have already read Handout B, which is my handout on the various number systems in use in the computer, then you recognize this number as a Hexadecimal number. But what exactly does it mean?

What is that funny number?

The format of this number may appear familiar to you. We saw in Handout B that the **&H** combination indicates that this number is a Hexadecimal numbers. The rest of the number I'll discuss in just a few moments. My point is that Visual Basic establishes a default value for the color property of your form and any control you place on the form. How does Visual Basic know what the default value should be? It looks for the user's color preference in their Control Panel. Naturally, the user's preferred color selections will vary from machine to machine. But the default value for the BackColor property won't change, as it just refers to the Control Panel value. Here's a screen shot from my Control Panel to show you what I mean.



The default value for the BackColor of the form is always **&H8000000F&,** although the actual color that this value represents may vary from computer to computer. According to the Visual Basic documentation, this number tells Visual Basic to look up the color that the user has specified in the Windows Control panel for the WindowBackground setting. For the ForeColor property of the form, again according to the Visual Basic documentation, Visual Basic uses the 'Window Text' color selection in the Windows Control Panel.

I did a little experiment, which you'll repeat yourself later, where I noted the BackColor and ForeColor properties of the Form and the controls that we'll discuss during our course. Here's a chart I have prepared based on that experiment.

Object	BackColor Property	ForeColor Property
CheckBox	&H8000000F&	&H80000012&
Command Button	&H8000000F&	None
Form	&H8000000F&	&H80000012&
Frame	&H8000000F&	&H80000012&
Image	None	None
Label	&H8000000F&	&H80000012&
ListBox	&H80000005&	&H80000008&
Option Button	&H8000000F&	&H80000012&
PictureBox	&H8000000F&	&H80000012&
Textbox	&H80000005&	&H80000008&

Now the natural question to ask at this point is, how do you know what these numbers correspond to in the Control Panel. That's what this chart tells you. I developed this chart by accessing the Visual Basic Object Browser.

System Color Constant	Hex Value	Description
vbScrollBars	&H80000000	Scroll bar color.
vbDesktop	&H80000001	Desktop color.
vbActiveTitleBar	&H80000002	Color of the title bar for the active window.
vbInactiveTitleBar	&H80000003	Color of the title bar for the inactive window.
vbMenuBar	&H80000004	Menu background color.
vbWindowBackground	&H80000005	Window background color.

vbWindowFrame	&H80000006	Window frame color.
vbMenuText	&H80000007	Color of text on menus.
vbWindowText	&H80000008	Color of text in windows.
vbTitleBarText	&H80000009	Color of text in caption, size box, and scroll arrow.
vbActiveBorder	&H8000000A	Border color of active window.
vbInactiveBorder	&H8000000B	Border color of inactive window.
vbApplicationWorkspace	&H8000000C	Background color of multiple-document interface (MDI) applications.
vbHighlight	&H8000000D	Background color of items selected in a control.
vbHighlightText	&H8000000E	Text color of items selected in a control.
vbButtonFace	&H8000000F	Color of shading on the face of command buttons.
vbButtonShadow	&H80000010	Color of shading on the edge of command buttons.

vbGrayText	&H80000011	Grayed (disabled) text.
vbButtonText	&H80000012	Text color on push buttons.
vbInactiveCaptionText	&H80000013	Color of text in an inactive caption.
vb3DHighlight	&H80000014	Highlight color for 3D display elements.
vb3DDKShadow	&H80000015	Darkest shadow color for 3D display elements.
vb3DLight	&H80000016	Second lightest of the 3D colors after vb3DHighlight.
vbInfoText	&H80000017	Color of text in ToolTips.
vbInfoBackground	&H80000018	Background color of ToolTips.

The chart shows the Visual Basic System Color Constant, along with its Hex value, and what the value means. I'll discuss Color Constants in just a few moments. For now, suffice to say that a Visual Basic Color Constant is just an easier way to remember a long Hexadecimal number, and its name can be used interchangeable with the value in Visual Basic code.

Note here that the value for the BackColor of the form (&H80000F) according to Visual Basic's documentation should correspond to the WindowBackground setting in the Control Panel. However, the Object Browser's description for the Color Constant indicates that this value is the setting for the user's selection for the color of a Command Button.

Pixels

Yes, you've seen and heard this term before. Pixels are the dots that form the characters and graphics displayed on your computer's monitors. Pixels are also a unit of measurement for computer monitors. For instance, your display monitor may display 800 pixels horizontally and 600 vertically. Pixels are varied in intensity and color to display the many beautiful colors that you see on your computer's monitor. A single pixel is capable of displaying over 16 million different colors by combining varying proportions of Red, Green and Blue (more on that later).

When color monitors were first introduced, there was a brief period of time when the colors on a display monitor were produced by combining three pixels. With today's modern technology, combining pixels in this way to form colors is not necessary, but this is something you should bear in mind if you are writing a program for a user with an older computer, and you choose a color other than the default color schemes.

If the user has older hardware (either a monitor of Video display adapter), it's possible that the PC will not be able to display all 16 million plus colors natively, that is, by using a single pixel. If that's the case, then the PC's hardware will attempt to produce the color by combining three pixels. This process is called 'dithering' and the results are less than optimal. The user will be able to tell the difference.

Those aren't the primary colors!

I mentioned that a pixel's color is created by mixing the proportion of Red, Green and Blue. But those aren't the so-called primary colors! You're correct. The primary colors of light are Red, Blue and Yellow. And if I remember my kindergarten training, green is actually the combination of Blue and Yellow. Why is it that computer engineers decided to use Red, Green and Blue to produce the 16 million colors of today's modern display monitors over the primary colors of Red, Blue and Yellow?

Well, as it turns out, the colors Red, Green and Blue are known as the **Additive Primary Colors.** And combining the Additive Primary Colors produces more colors than combining the Primary Colors of Red, Blue and Yellow. As it turns out, it isn't only the computer engineers who know this. The next time you go the theater, take note of the colors used for stage lighting. You'll see the Additive Primary colors in use.

But What's that number in the BackColor property mean then?

Let's get back to that default number in the BackColor property of the form.

&H8000000F&

This number you see here serves a dual function. Sometimes it represents the Visual Basic Color Constant I referred to earlier. And sometimes it represents a Hexadecimal value that corresponds directly to one of the 16 million possible colors that can be displayed on a color monitor. How can you tell what's what? If the first four characters of the number start with anything other than &H00, then you know you are looking at a Visual Basic Color Constant. If the number starts with &H00, then the remaining 6 Hexadecimal characters represent one of 16 million colors.

That's right. With just 6 Hexadecimal characters, 16 million colors (actually 16,777,216 colors) can be displayed. As I mentioned earlier, these colors can be displayed by varying the intensities of Red, Green, or Blue. 2 Hex characters represent the intensities of Red, Green and Blue, and their values range from 0 to 255 Decimal, or 0 to FF in Hex. An example will make this easier to understand.

For instance, the hex value

&H000000FF&

represents the color Red. Read the Red-Green-Blue values from Right to Left. In other words, “FF” is the Red component of the color. '00' is the Green component. And '00' is the Blue component. The displayed color will be Red because the highest intensity of Red has been selected.

This hex value

&H0000FF00&

represents the color Green. Again, reading the Red-Green-Blue values from right to left, '00' is the Red component, 'FF' is the Green component, and '00' is the Blue component.

This hex value

&H00FF0000&

represents the color Blue. Again, reading the Red-Green-Blue values from right to left, '00' is the Red component, '00' is the Green component, and ' FF ' is the Blue component.

What about this hex value?

&H00FFFFFF&

This value represents the color White. White is the full intensities of Red, Green and Blue. Reading from Right to Left, we have the maximum value('FF')for Red, Green and Blue.

My favorite color, Cyan, is achieved by combining the full intensities of Green and Blue, with no Red.

&H00FFFF00&

The important thing to note about all of these examples is that the Hex value began with &H00. If you see a Hex value begin with &H80, then you know that you are dealing with a Color Constant. Here are Hexadecimal equivalents for some common colors.

Property Value	Color
&H00000000&	Black
&H00FF0000&	Blue
&H00FFFF00&	Cyan
&H0000FF00&	Green
&H00FF00FF&	Magenta
&H000000FF&	Red

&H00FFFFFF&	White
&H0000FFFF&	Yellow

What does all of that mean?

Where to from here? I'd like to suggest that you take a moment now to experiment with the Visual Basic Default colors on your own system.

1. Start a new Visual Basic Standard.EXE project.

2. Use your toolbox to place a Command Button, Checkbox, Frame, Image, Label, ListBox, OptionButton, PictureBox and Textbox on the form.

3. Bring up the Properties Window, and record the default values for the BackColor and ForeColor properties for the form and each control.

Here are the default values that I noted:

Object	**BackColor Property**	**ForeColor Property**
CheckBox	&H8000000F&	&H80000012&
Command Button	&H8000000F&	Has no ForeColor property
Form	&H8000000F&	&H80000012&
Frame	&H8000000F&	&H80000012&
Image	None	None
Label	&H8000000F&	&H80000012&

ListBox	&H80000005&	&H80000008&
Option Button	&H8000000F&	&H80000012&
PictureBox	&H8000000F&	&H80000012&
Textbox	&H80000005&	&H80000008&

What does all of this mean? It means that without any effort on your part, you have default colors already selected for you, and that you already have a 'leg up' with the user since these are colors that they have already selected in the control panel. Bear in mind, that if you accept these default values, if the user goes to the control panel and changes their color selections, the colors in your program will also change automatically. Who could ask for more?

Well, to put it bluntly, programmers. For whatever reason, programmers love to deviate from the default colors. And so they customize. And some even permit the user to change their own colors. So yes, if you want to get fancy, you can change the default color selection in your program. All you need to do is provide Visual Basic with a valid color value, either in the Properties Window, or through Visual Basic code. And that gets us back to that Hexadecimal number notation again.

What's that you say? Hex still scares you a bit? I don't blame you. Suppose all you want to do is change the BackColor property of the form to Cyan. It's difficult to remember that the Hex value for Cyan is &H00FFFF00&. Surely there must be an easier way? And there is. In fact, there are several easier ways to change colors in Visual Basic.

Color Constants

The first alternative to those nasty Hexadecimal values is to use a Visual Basic's Color Constant, not to be confused with the Visual Basic System Color Constants we discussed earlier in this handout. Both the SystemColor Constants and the Color Constants are a subset of the much larger universe of Visual Basic Intrinsic Constants. The idea behind the Visual Basic Intrinsic Constants is to give you a name that you can remember in place of a hard to remember value. For instance, what's the Hex value of the color cyan? Even though I

told you that it's my favorite color, off the top of my head I don't remember its Hex value. But it's easy to remember its Color Constant name. It's vbCyan. Get the point?

Here is a list of the Color-related Visual Basic Intrinsic Constants. To view these yourself, select On-Line Help, then select Constants, then scroll down to Color.

Constant	Value	Description
vbBlack	&H0	Black.
vbRed	&HFF	Red
vbGreen	&HFF00	Green
vbYellow	&HFFFF	Yellow
vbBlue	&HFF0000	Blue
vbMagenta	&HFF00FF	Magenta
vbCyan	&HFFFF00	Cyan
vbWhite	&HFFFFFF	White

Here's an Exercise show you how easy using an Intrinsic Constant to change the color of the form can be.

1. Start a new Visual Basic Standard.EXE project.

2. Use your Toolbox to place a Command Button on the form. Accept the default name that Visual Basic assigns.

3. Double click on the Command Button, and place the following code into the Click Event Procedure of the Command Button.

```
Private Sub Command1_Click()
```

```
Form1.BackColor = vbRed

End Sub
```

4. Run the program. There is no need to save either the project or the form for these non-China Shop project related Try-It-Outs, so when Visual Basic asks you if you want to save changes to those files, answer 'No' by clicking on the 'No' button.

5. Click on the Command Button. The color of the form will change to red.

You can only use Intrinsic Constants in Visual Basic code. The Properties Window only accepts numbers.

The QBColor Function

One problem with the Color Constants is that there are only eight of them. While this is fine for color-challenged people like me, many programmers feel the need to work with more than eight colors. Fortunately, there are two other methods to specify colors--- the QBColor Function, and the RGB Function.

The QBColor function is a “holdover” from the original version of Basic. The QBColor expands on the eight colors provided by the Color constants and gives you sixteen. The QBColor function requires a single argument, which is a number ranging from 0 to 15.

The color argument has these values:

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Light Blue

2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Yellow	14	Light Yellow
7	White	15	Bright White

The QBColor function returns a value that can then be assigned to any of the color properties. For instance, this code returns a value which when assigned to the BackColor property of the form changes it to Green.

Form1.BackColor = QBColor(2)

Again, the advantage of the QBColor function over using the Intrinsic Color Constants is the eight additional colors that it can generate for you. The disadvantage of the QBColor function is that you need to know the argument necessary to produce that color, and that can be a real problem.

The RGB Function

The RGB function , while not extremely user friendly, gives you the ability to generate any valid color property value.

Here is the syntax for this function.

RGB(Red,Green,Blue)

As you can see, the RGB function requires three arguments, with valid values for each being 0 to 255. Similar to what we saw earlier when examining the syntax of the hex value for a color property, each argument represents varying intensities of Red, Green, and Blue.

The following table lists some standard colors and the red, green, and blue values they include:

Color	Red Value	Green Value	Blue Value
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Cyan	0	255	255
Red	255	0	0
Magenta	255	0	255
Yellow	255	255	0
White	255	255	255

For instance, using the RGB function, this code will change the BackColor of Form1 to Red.

```
Form1.Backcolor = RGB(255,0,0)
```

Since these arguments are named arguments, you could also use this syntax. This code will change the BackColor of Form1 to Green.

```
Form1.BackColor = RGB(Red:=0, Green:=255, Blue:=0)
```