CHAPTER 4

Color

Since humanity began, we have been using the intrinsic colors of objects in the natural world. The artists of Altamira and Lascaux used ochres, colored earths, to draw their magnificent animals some 20,000 years ago. A now-extinct shellfish provided the dye used for the purple stripe on a Roman senator's toga. The blue robes of Sandro Botticelli's gentle Madonnas are tinted with ground lapis lazuli, a blue stone.

In this chapter, we move from typography to color theory—color's characteristics and interactions. Ultimately, of course, color is all about light, and the way our eyes react to it. The human eye can distinguish upwards of ten million distinct colors—a huge problem space that any systems analyst will tell you must be structured and organized.

This is what color theory does. However, traditional color theory is based on the mixture of pigments, while the display of color on a computer monitor is a mixture of light, which, as we'll see, is slightly different in its details.

The first two sections of this chapter deal with traditional, pigment-based color theory without translation. In designing color for your user interfaces, it's traditional theory that must guide you. The next section looks at the differences between color models, including the ARGB model used on computers, and finally, the last section looks at the .NET Framework objects used to manipulate color.

Come on, admit it, it's a lot more fun than memory allocation models (or at least no more tedious).

Understanding Color

Like the other graphic elements that we're examining in this first part of the book, there's a set of terms that folks in the field use to talk about color, and we'll examine them here.

The Dimensions of Color

Every color can be defined along three dimensions: hue, saturation, and value. **Hue** is what we normally think of as the color itself—red, yellow, or puce. **Value** is the relative lightness or darkness of a color—pink or magenta.

Saturation is measure of intensity, which is a function of purity of hue. If you mix equal amounts of the three primary colors, the result is gray. Imagine a gradient with a pure hue on one end and gray on the other—that's a measure of saturation.

The color wheel is a useful starting point for working with color. There are actually quite a few different configurations of color wheels, and we'll look at a couple others in this chapter, but the one shown in Figure 4–1 in the color insert, is the most common.

The three colors that cannot be created by combining other colors—yellow, red, and blue—are the **primary colors**. **Secondary colors**, created by mixing two primaries, are placed between them on the wheel. For example, green is the mixture of blue and yellow.

As shown in Figure 4–2 in the color insert, the wheel can be further refined by including the **tertiary colors** such as red-orange or blue-green, which are mixtures of a primary and secondary color.

You may be thinking that there are some colors missing from the color wheel—black, white, gray, and brown, often called the **neutrals**. In fact, black and white aren't technically colors. If you think of light, of course, black is its absence and white its presence.

You might expect that mixing the three primary colors would result in white. In reality, the result is gray. (You can also get gray by mixing black and white pigments, of course, because neither color, *as a pigment*, is a pure color.) Black and white are effectively primaries because they can't be mixed from other pigments.

They can, of course, be mixed with other pigments. When you mix a pigment with white, the result is a **tint**. When you mix a pigment with black, the result is a **shade**. If you add both black and white to a pigment, the result is a **tone**.

The other neutral is brown. Every time I've taught color mixing, somebody asks how you get brown. It *is* difficult to look at the color wheel and see how you could possibly arrive at such a distinctive color. So, here's the secret: Brown is a shade of orange, while beige is its tone.

Colors have one more characteristic that deserves mentioning: They are either warm or cold. The warm colors, as shown in Figure 4–3 in the color insert, are in the red-orange range, while cool colors are in the blue-green range.

Conventional wisdom has it that warm colors advance while cool colors recede, and while there's truth to that, it tends to be more important when you're painting walls than designing user interfaces. There's a big difference between the effect of an 8-foot wall and an 8-inch screen.

Color Harmonies

There is, of course, no limit to the number of effective color combinations, but that knowledge isn't much help when you're faced with designing a color scheme. Not to fear; there are traditional systems to help you organize your work.

The physical relationship between colors on the color wheel is the basis of traditional color systems, called **color harmonies**. Color harmonies are often referred to as color schemes, but that's not correct. A **color scheme** is simply the colors used in a project. A color harmony complies with one of the relationships described here.

A **monochromatic** color harmony uses a single color, perhaps with the addition of black and white. Monochromatic harmonies are obviously the simplest to implement. Pick a color you like, use its tints, shades, and tones, and away you go.

Invisible color schemes can be a good thing.

Like a document that uses a single typeface, monochromatic schemes are generally safe, slightly formal, and can be invisible. Sounds boring, but if you're building a data entry system, invisible is a good thing.

Analogous color harmonies are also fairly easy to use, but a bit more vibrant. As shown in Figure 4–4 in the color insert, an analogous harmony uses a primary hue along with one of its secondaries—blue and green in the example shown. Analogous harmonies may also include the intermediate colors—blue-green in the example.

Analogous color schemes can be beautiful, but you must be careful to maintain enough contrast. Green text on a blue background is going to be pretty close to unreadable. But, dark green text on a pale blue background can be quite pleasant.

Make sure you have enough contrast when choosing an analogous color scheme.

A **triadic color harmony**, shown in Figure 4–5 in the color insert, combines three equidistant colors, either the three primaries, the three secondaries, or three tertiaries.

Triadic color harmonies tend to be quite vibrant, even if you use pale or unsaturated versions of your hues. To use a triadic harmony successfully, you need to balance the three colors very carefully, using only small amounts of two of the colors, for example.

Complementary colors are directly opposite each other on the color wheel, as shown in Figure 4–6 in the color insert. Complementary colors have a special relationship. You've probably done the exercise—stare at a square of green for a few seconds and then look at a white wall. You'll see an after-image of red. Red and green are, of course, complementary colors.

Balance triadic harmonies carefully.

If you put two complementary colors side by side, the edges tend to vibrate. So, for example, if you choose a text color that's the complementary of the window background, the text will start to move around on the screen. (Try it. It'll make you homesick for the sixties.)

Use your main color's complement as an attention-getter.

Complementary colors are tricky to use in large doses, but using a color's complement is an excellent technique for drawing attention to an area of the screen.

Complementary color harmonies can also work if you contrast the value and intensity of the two colors. Maroon and pale green, for example, are a classic Art Deco combination, and can work even for text and background, at least in relatively small doses.

The split complementary harmony is a good choice for polychrome schemes because it's so hard to mess up.

The final classic color harmony is the **split complementary**, which combines one color with the two colors on either side of its complementary, as shown in Figure 4–7 in the color insert.

The split complementary harmony sounds tricky, but is actually difficult to mess up, which makes it a good, safe choice for polychrome color schemes.

The split complementary is one of my favorite harmonies. A color scheme based on the split complementary *is* colorful, but hardly ever loud. It might not be the best choice for straight data entry applications, where efficiency is more important than appearance, but remember that any color harmony can combine black and white for its text display.

Using Color

A lot of the basic principles of using color are implicit in the color harmonies we've just examined—these color harmonies represent basic organizing principles.

Beyond the harmonies, one begins to enter areas of aesthetics that are, unfortunately, outside our scope. Is red sexy? Is blue calming? Yeah, well, sometimes. And sometimes pink is romantic, and sometimes it's funky.

Realistically, the only useful advice I can give you is to look at stuff. Art galleries are wonderful, of course, but magazines, and restaurants, and the people you see walking down the street are also sources of color ideas.

Most importantly, if you work for a company, look at their marketing materials. Even small companies probably have brochures, or at least business cards, and the colors used there are de facto standards for the company. Think of IBM blue, UPS brown, or Martha Stewart green.

To learn to use color, look at stuff.

In choosing your color scheme, you should also remember the basic principles we studied in Chapter 1, *Interface Design*. Alignment and proximity don't really apply, of course, but contrast, consistency, and focus and flow certainly do.

At the beginning of this chapter we talked about the three dimensions of color: hue, saturation, and value. Any combination of these qualities can be used to establish contrast.

Obviously, you must have sufficient contrast between elements for visual clarity. That said, color schemes with greater contrast, or contrast in multiple dimensions, tend to be brighter and more casual, while color schemes with less contrast tend to be more formal. There are exceptions, of course, but think of a page in several bright primary colors compared to one with, for example, navy blue type against a pale blue background.

Contrast in color is one of the two primary methods for controlling focus and flow. (The other is size.) The one red item on an otherwise beige page will invariably have the most visual weight.

Red is often used to indicate danger, and we're conditioned to pay attention to it. But that principle has less weight on a computer screen than it does in the physical world—the single beige item on an otherwise red page will also be the first to draw the eye. It's the contrast that attracts the viewer's attention, not the color's conventional meaning.

Red is one of the best examples of conventional color uses. Green and yellow are others. The conventional meanings of these colors—stop, go, caution—are constantly reinforced. As always, it's best to comply with the convention unless you have compelling reason not to.

There's a reason warning icons are yellow. Make yours purple, and you're just going to confuse people. The exception, of course, is if your color scheme is predominantly yellow. In that case, purple's not a bad choice, since it's yellow's complement, and thus stands out strongly. (But I'd probably use red, myself.)

One final note on using color: Color is useful for distinguishing things, but not particularly good for coding them. We all do it of course, and color-coding doesn't do any harm; just don't rely on it exclusively. In the first place, a surprising number of people are color-blind. But even among those who can distinguish colors easily, you can't just assume that people are going to remember that the green Contact Details screen is used for customers while the otherwise-identical Contact Details screen is used for employees. (Yep, I've seen it.)

Think about it—when you were a baby, somebody (presumably) taught you that red things were often hot. But unless you're a prodigy, it's unlikely you got it the first time up, and I'd be willing to bet that you've burned yourself once or twice since then. And "red means ouch" is a lot more compelling than "yellow means employee." So use multiple cues—color by itself is insufficient.

Color Models

Color is a very slippery thing to quantify, but of course we must quantify it if we are to work in the digital world. There are three different models in common usage, and we'll examine each in this section. But ultimately, all three models are derived from the physics of color, and so we'll begin our examination of color models with a little (very little) of that.

Additive and Subtractive Color

You've probably seen the results of shining light through a prism—the white light is split into its component colors, the **spectrum**. You may recall that what's happening here is that each color has a different wave length.

All color is a function of the way our eyes perceive different wave lengths of light. When you look at a hyacinth, you see it as white because the petals reflect all wave lengths of light back to you. You see the leaves as green because they absorb all wave lengths *except* those our eyes perceive as green. Pigments work exactly the same way—titanium white reflects all wave lengths, while lamp black absorbs them.

NOTE: Technically, pigments are never pure in the sense that physicists use the term. Titanium white reflects *almost* all wave lengths of light, but not all of them. If you doubt this, just walk into the nearest paint store. There are dozens of whites, each of which absorbs a slightly different range of wave lengths. But the model is close enough for our purposes.

Now, the important thing to understand here is that if you're looking at anything that doesn't actually glow, its color is determined by the color that's *absorbed*. Yellow crocus absorbs all the red, green, and blue wave lengths. Purple iris absorbs the red, yellow, blue, and green.

When you combine colors in the form of paint or pigment, you're not so much adding colors together as subtracting them. Blue and yellow make green because the blue absorbs some of the yellow, the yellow absorbs some of the blue, they both absorb red, and what's left over is green.

I do understand that this particular bit of knowledge is singularly unhelpful when you're up to your elbows in finger paint. But it does explain why the color mixtures that are represented on the traditional color wheel are called the **subtractive** model.

The inverse of the subtractive model is the additive model. While the subtractive model explains what happens when you combine things that don't emit light, the **additive** model explains the interaction of light of various wave lengths—the combination of colored light. Combining colored light is, of course, precisely what you're doing when you specify color on a computer screen.

I suppose because the ability to combine colored light is fairly new, most color theory is based on the subtractive model, and that's what we've been examining so far in this chapter. The additive model has a different color wheel, as shown in Figure 4–8 in the color insert.

Use the subtractive color wheel for designing color schemes.

The primary colors on the additive color wheel are red, green, and blue rather than red, yellow, and blue. Red and green, for example, combine to make yellow. (I don't think I shall ever get used to that.)

Notice that the relationships between colors aren't the same on the additive color wheel. Yellow and blue are directly opposite each other on the wheel, but they aren't complements, and they don't behave the same way in relationship to one another. When you're working out color schemes, work from the subtractive color wheel.

HSB Color

Hue, saturation, and value are the dimensions used when we talk and think about color. The additive and subtractive models, and the color wheels that represent them, are used when we physically manipulate color. The HSB color model sits somewhere between the two.

The **HSB model** specifies color as a combination of three values: hue, saturation, and brightness. Hue and saturation correspond directly to the hue and saturation of color theory, while brightness corresponds roughly to value. The difference between brightness and value is the difference between the additive and subtractive models—a lack of light isn't so much black as, well, dark.

In the HSB model, hue is measured as degrees on the color wheel, with pure red being both 0 and 360 degrees

Saturation and brightness are both measured as percentages in the HSB model. For example, a saturation value of 0 is white, while 100 is the fully saturated color.

As we'll see, the .NET Framework provides only limited support for the HSB model, and in fact it is not as widely used as the next two models.

CMYK Color

The color model most often used for printing is **CMYK**, which stands for ***C***yan-***M***agenta-***Y***ellow-blac***K***. (I *think* the K is to avoid confusion between the "b" in black and the one in blue, but I wouldn't swear to that.)

All four values in the CMYK are expressed as percentages, which translate directly to the percentage of the corresponding color ink required to reproduce the color in the four-color printing process.

Because of its importance in the printing process, the CMYK model is widely supported in drawing and publishing software, but surprisingly, isn't supported by the .NET Framework.

RGB and ARGB Color

The final common color model is RGB, which represents the additive primaries ***R***ed, ***G***reen, and ***B***lue. This is the standard model used for specifying color for computers.

The .NET Framework adds an additional value to red, green, and blue: the alpha value. The **alpha value** specifies the transparency of the color, the extent to which the color is blended with its background. Like the other values in the RGB model, alpha values are specified as an integer between 0 (transparent) and 255 (fully opaque).

NOTE: By convention, ARGB values are specified in hexadecimal, making the range 0x00 to 0xFF.

When the alpha value is less than 0x00, the actual color of each pixel is determined by the following formula:

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

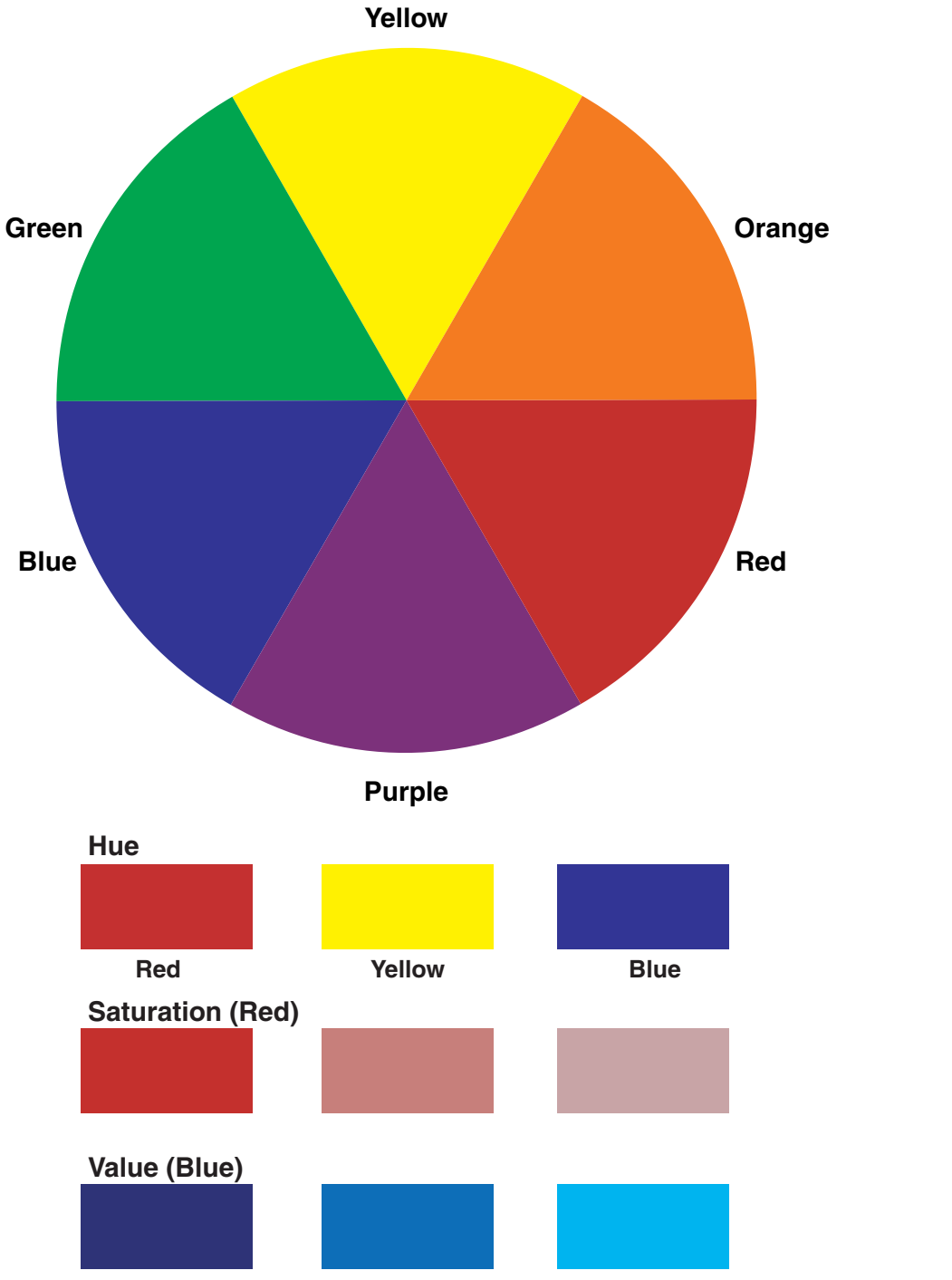


Figure 4–1 Basic Color Wheel

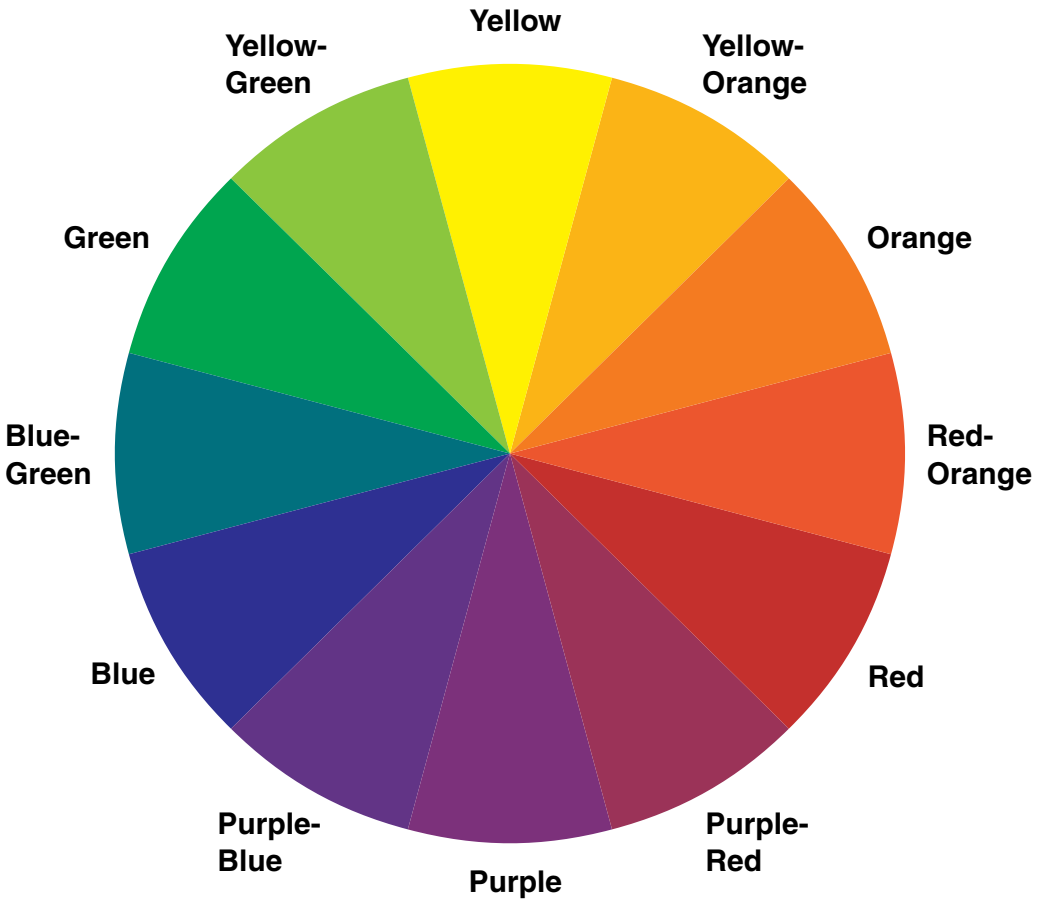


Figure 4–2 Tertiary Color Wheel

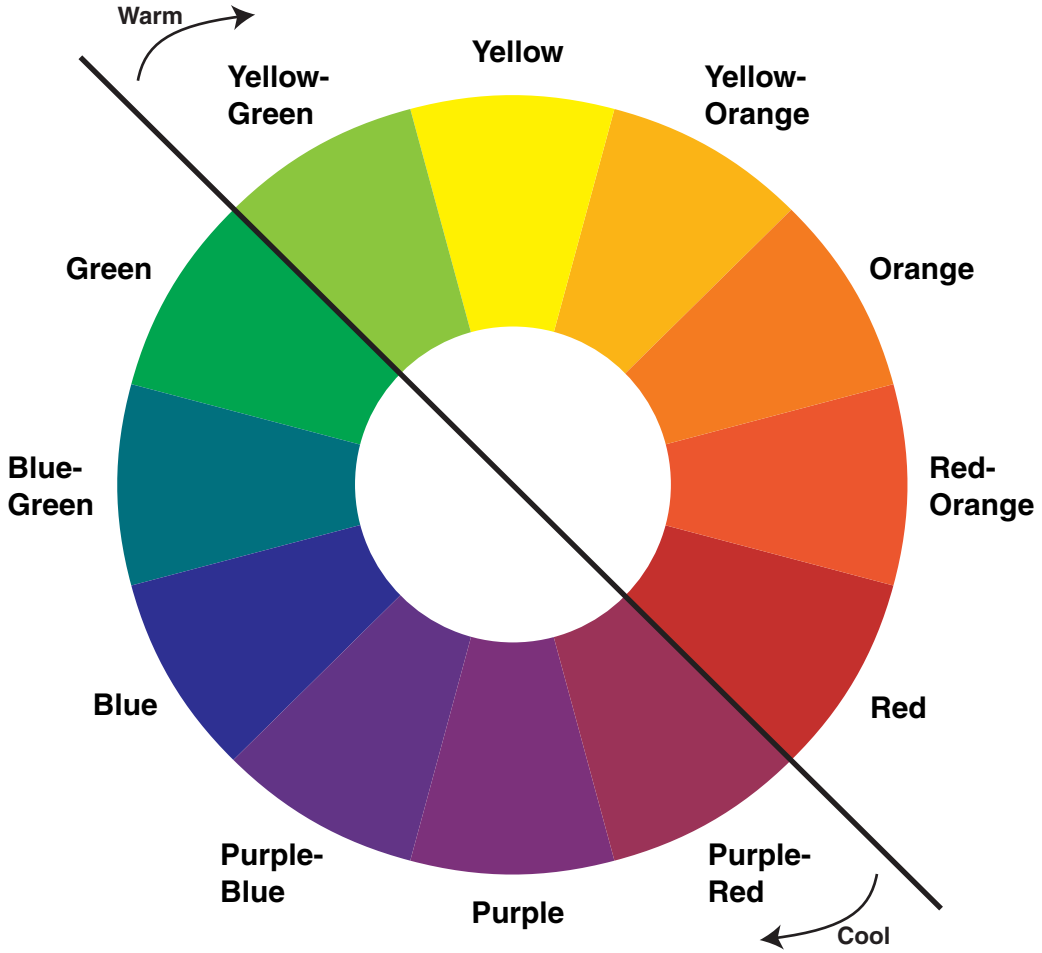
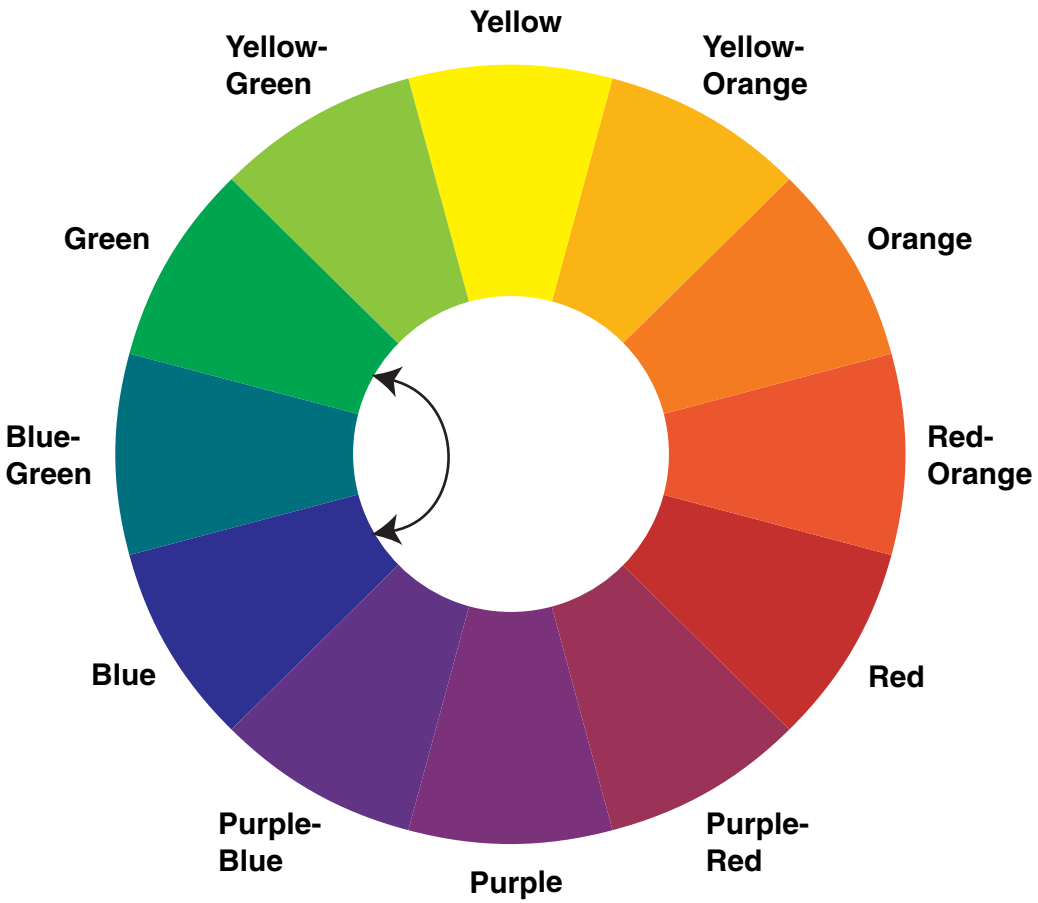


Figure 4–3 Warm and Cold Colors



This is really difficult to read.

This is much easier to read.

Figure 4–4 An Analogous Color Harmony

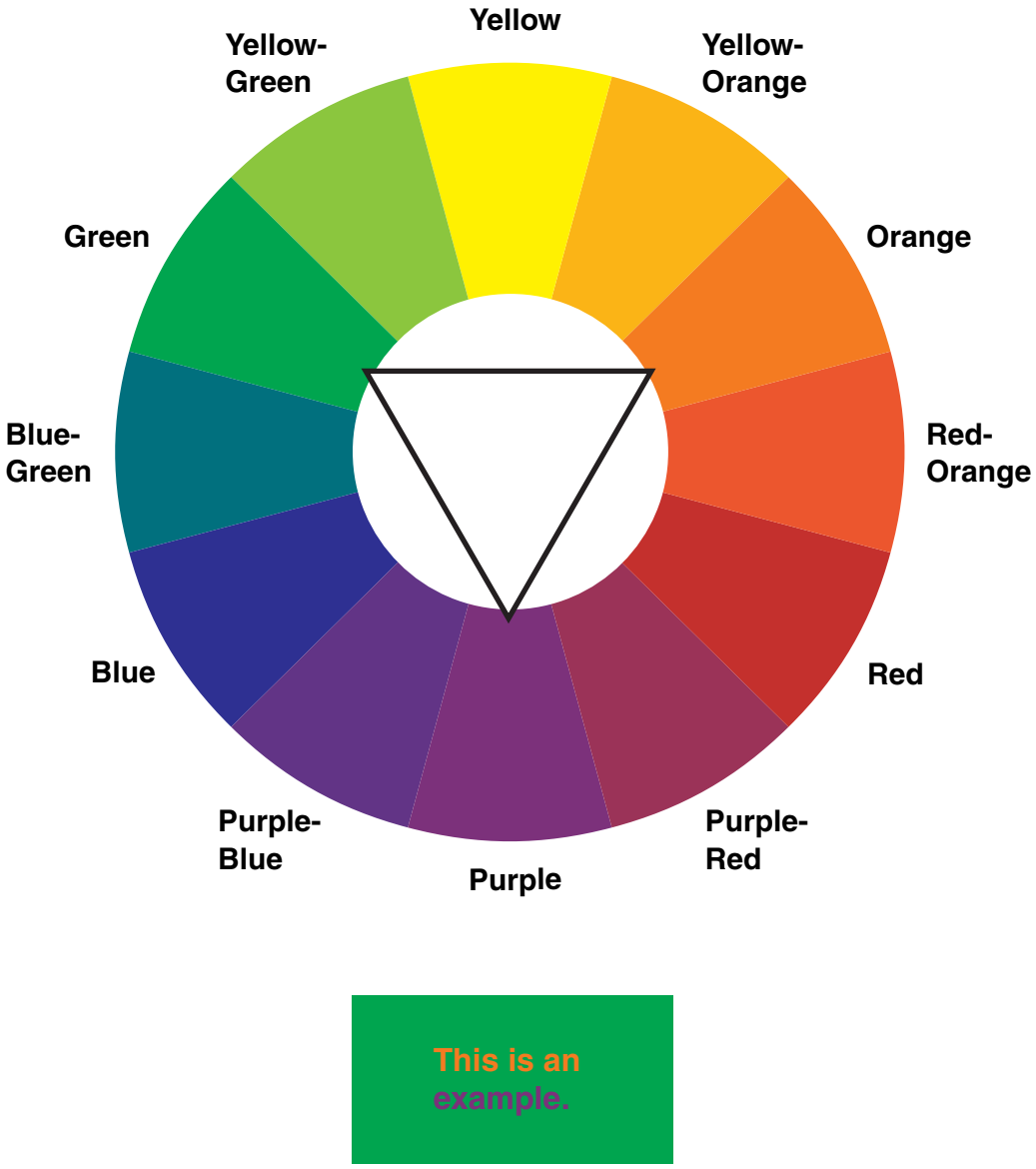


Figure 4–5 A Triadic Color Harmony

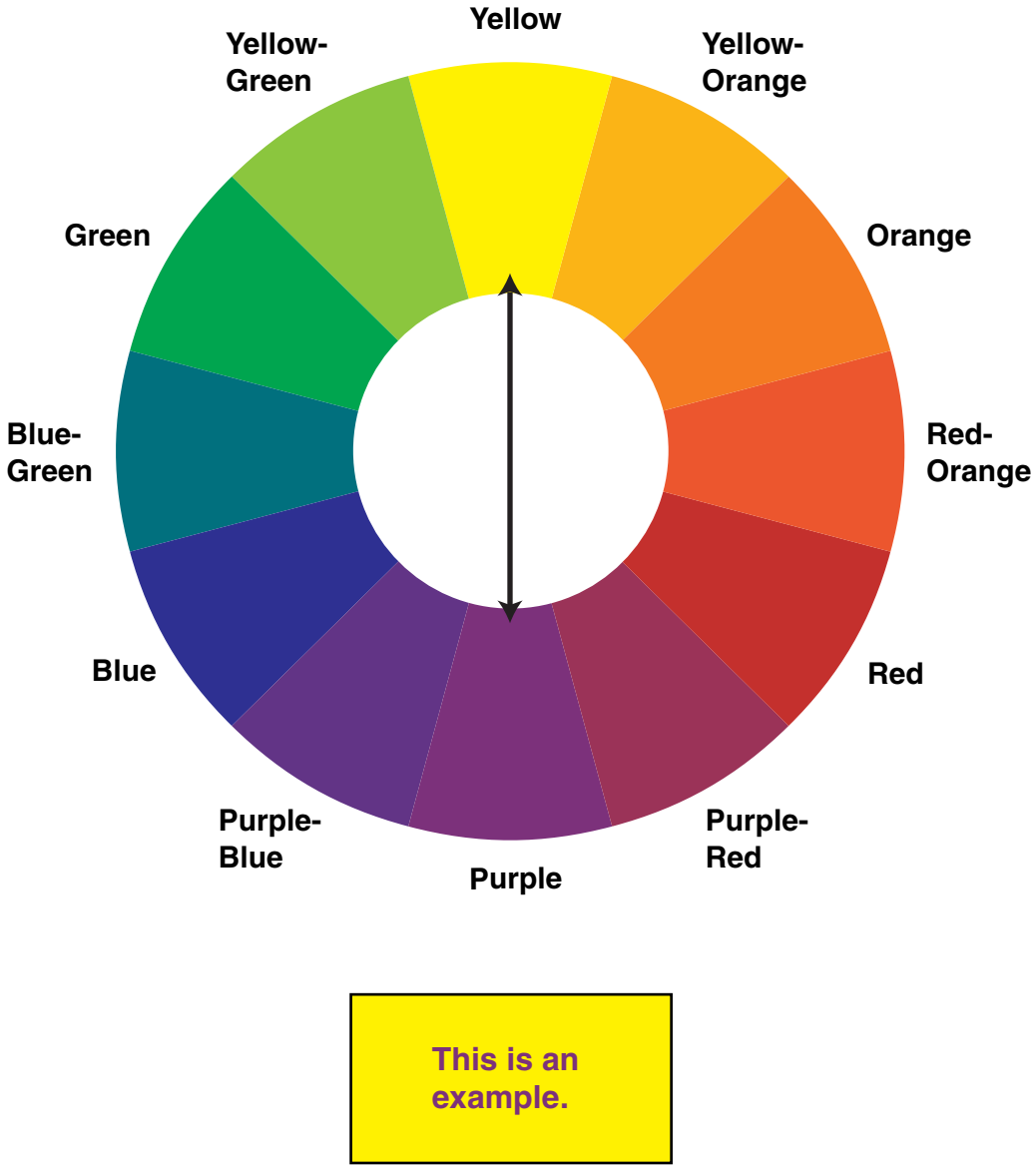


Figure 4–6 Complementary Colors

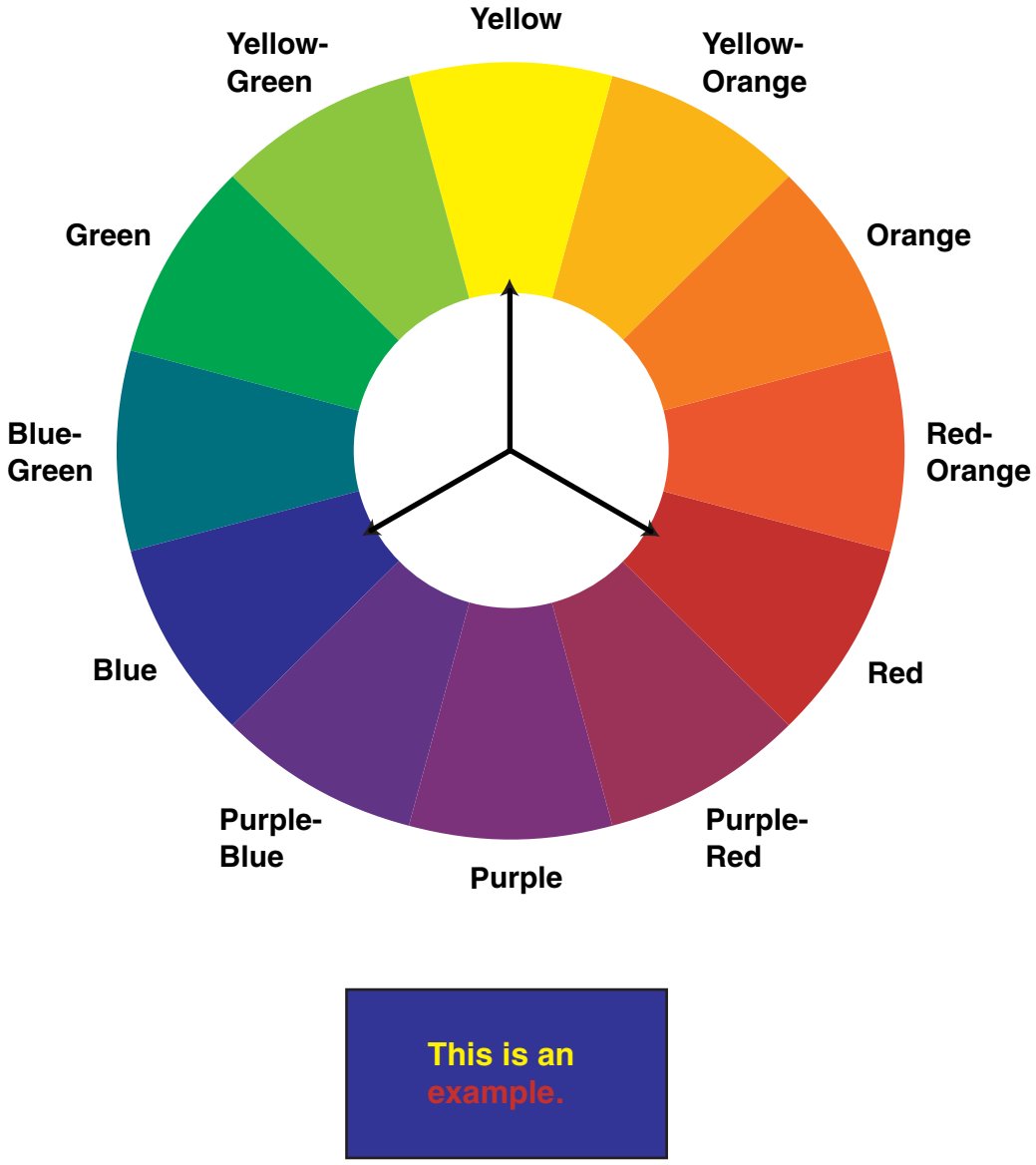


Figure 4–7 Split Complementary Color Harmony

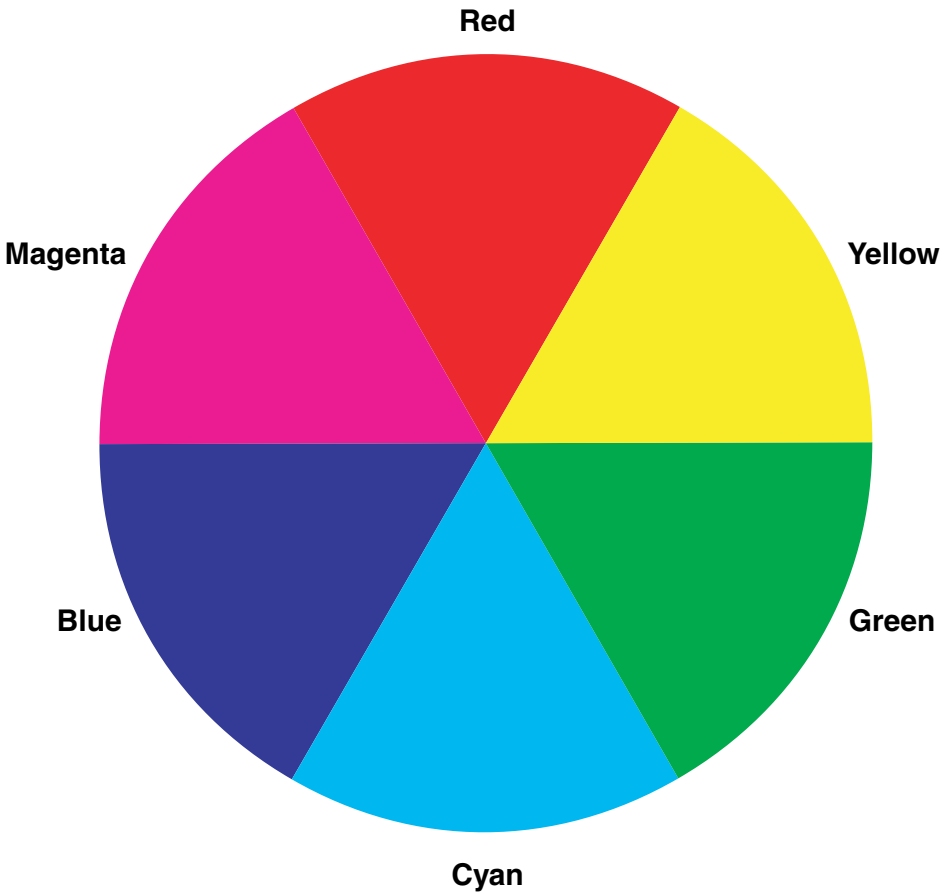


Figure 4–8 The Additive Color Wheel

Fortunately, GDI+ takes care of the calculations for you. You need only specify the alpha value. (It's easiest to think of alpha as a percentage value, although that does require a little arithmetic translation to arrive at the hex value.)

There's an ARGB exerciser included in the sample code of this chapter, so you can play with alpha values to get a feel for them.

Color in the .NET Framework

As you'd expect, color is represented by an object in the .NET Framework; in this case, a structure rather than a class, the Color structure in the System.Drawing namespace.

The Color structure is supported by the KnownColor enumeration and the SystemColor and ColorTranslator classes, all also part of System.Drawing.

The Color Structure

The Color structure exposes three static methods that function as pseudo-constructors, as shown in Table 4–1.

The FromArgb method is overloaded, exposing four different versions that allow you to specify various combinations of the ARGB component values. The FromKnownColor and FromName versions create a Color structure based on the KnownColor enumeration.

As shown in Listing 4–1, you pass the enumeration member directly to the FromKnownColor method, and the name of the color, as a string, to the FromName method.

Table 4–1 Color Pseudo-Constructors

Method	Description
FromArgb	Creates a Color structure from its ARGB component values
FromKnownColor	Creates a Color structure representing the specified member of the KnownColor enumeration
FromName	Creates a Color structure representing the member of the KnownColor enumeration specified its name (passed as a String)

Listing 4–1 Two methods for creating a Color structure based on the KnownColor enumeration

```
Dim clrEnum As Color, clrName As Color
clrEnum = Color.FromKnownColor(KnownColor.Beige)
clrName = Color.FromName("Beige")
```

In addition to the pseudo-constructors, the Color structure exposes five methods that are useful for translating between various color models, as shown in Table 4–2.

NOTE: You can't create a Color directly from the HSB model in the .NET Framework, although as we'll see, the Visual Studio color picker allows it.

In addition to its methods, the Color structure exposes two sets of properties. The first set, shown in Table 4–3, provides information regarding the Color.

The A, R, G, and B properties should be self-explanatory, as should the Name property.

The IsKnownColor and IsNamedColor properties, which always return the same value, indicate whether the Color structure was created from a KnownColor using either the FromKnownColor or FromName method. These properties will *not* return true if the ARGB value of a Color happens

Table 4–2 Color Structure Translation Methods

Method	Description
GetHue	Returns the HSB hue in degrees
GetSaturation	Returns the HSB saturation percentage as a value between 0 and 1
GetBrightness	Returns the HSB brightness percentage as a value between 0 and 1
ToArgb	Returns the 32-bit ARGB value representing the color
ToKnownColor	Returns a member of the KnownColor enumeration if one was used to create the color; otherwise, returns zero

Table 4–3 Color Structure Properties

Properties	Description
A	Alpha component value
R	Red component value
G	Green component value
B	Blue component value
IsKnownColor	Indicates whether the color was created from the KnownColor enumeration
IsNamedColor	Indicates whether the color was created from the KnownColor enumeration
IsSystemColor	Indicates whether the color was created from one of the first 26 members of the KnownColor enumeration
Name	The name of a known color

to match that of a member of the KnownColor enumeration—they don't perform a search or compare ARGB values.

The IsSystemColor property indicates whether the Color is one of the colors set by the user in the control panel. As we'll see in the next section, SystemColors are represented by the first 26 members of the KnownColor enumeration.

The sample program for this chapter includes a form that displays each of these properties for the KnownColors.

The other set of Color properties match the values of the KnownColor enumeration. These static properties allow you to quickly create an instance of one of the known colors.

Listing 4–2, a reprise of Listing 4–1, uses the static Beige property to create a third instance of the Color structure.

Listing 4–2 Using a static property to obtain an instance of a Color structure

```
Dim clrEnum As Color, clrName As Color, clrStatic as Color

clrEnum = Color.FromKnownColor(KnownColor.Beige)
clrName = Color.FromName("Beige")
clrStatic = Color.Beige
```

Known and System Colors

The .NET Framework supports a set of 140 standard colors through its KnownColor enumeration (and the related static properties of the Color structure).

These colors, which are listed in Table 4–4, were proposed, but not included, in the CSS standard, although they do not coincide with the so-called Web Safe color range. Despite this, they do represent a de facto HTML standard, and they're supported by recent versions of both of the two main browsers, Internet Explorer and Netscape Navigator.

Table 4–4 The KnownColor Enumeration

AliceBlue	DarkSlateGray	LightSalmon	PaleVioletRed
AntiqueWhite	DarkTurquoise	LightSeaGreen	PapayaWhip
Aqua	DarkViolet	LightSkyBlue	PeachPuff
Aquamarine	DeepPink	LightSlateGray	Peru
Azure	DeepSkyBlue	LightSteelBlue	Pink
Beige	DimGray	LightYellow	Plum
Bisque	DodgerBlue	Lime	PowderBlue
Black	Firebrick	LimeGreen	Purple
BlanchedAlmond	FloralWhite	Linen	Red
Blue	ForestGreen	Magenta	RosyBrown
BlueViolet	Fuschia	Maroon	RoyalBlue
Brown	Gainsboro	MediumAquamarine	SaddleBrown
BurlyWood	GhostWhite	MediumBlue	Salmon
CadetBrown	Gold	MediumOrchid	SandyBrown
Chartreuse	Goldenrod	MediumPurple	SeaGreen
Chocolate	Gray	MediumSeaGreen	SeaShell
Coral	Green	MediumSlateBlue	Sienna
CornflowerBlue	GreenYellow	MediumSpringGreen	Silver
Cornsilk	Honeydew	MediumTurquoise	SkyBlue
Crimson	HotPink	MediumVioletRed	SlateGray
Cyan	IndianRed	MidnightBlue	Snow
DarkBlue	Indigo	MintCream	SpringGreen
DarkCyan	Ivory	MistyRose	SteelBlue
DarkGoldenrod	Khaki	Moccasin	Tan
DarkGrey	Lavender	NavajoWhite	Teal
DarkGreen	LavenderBlush	Navy	Thistle
DarkKhaki	LawnGreen	OldLace	Tomato
DarkMagenta	LemonChiffon	Olive	Transparent

Table 4–4 The KnownColor Enumeration *(Continued)*

DarkOliveGreen	LightBlue	OliveDrab	Turquoise
DarkOrange	LightCoral	Orange	Violet
DarkOrchid	LightCyan	OrangeRed	Wheat
DarkRed	LightGoldenrodYellow	Orchid	White
DarkSalmon	LightGray	PaleGoldenRod	WhiteSmoke
DarkSeaGreen	LightGreen	PaleGreen	Yellow
DarkSlateBlue	LightPink	PaleTurquoise	YellowGreen

While it must be said that some of these colors are spectacularly ugly, the .NET Framework does make them easy to access. For example, Listing 4–3, excerpted from the chapter's example program, demonstrates how to read the names of the KnownColor enumeration into an array, and then display that list in a ListBox control.

Listing 4–3 Reading the names of the KnownColor enumeration into an array

```
Dim theNamedColors() as String

theNamedColors = System.Enum.GetNames(GetType(KnownColor))

Me.theListBox.Items.AddRange(theNamedColors)
```

The KnownColor enumeration also includes 26 colors (in positions 0 to 25) that correspond to the static properties of the SystemColors class, shown in Table 4–5.

Like SystemPens and SystemBrushes, the System Colors class represents the ambient properties set by the user in the control panel.

Table 4–5 SystemColors Properties

Property	Description
ActiveBorder	The color of the active window's border
ActiveCaption	The color of the background of the active window's title bar
ActiveCaptionText	The color of the text in the active window's title bar
AppWorkspace	The color of the background of the application workspace (the area in a MDI view that is not occupied by documents)

Table 4–5 SystemColors Properties *(Continued)*

Property	Description
Control	The color of the face color of a 3-D element
ControlDark	The color of the shadow color of a 3-D element
ControlDarkDark	The color of the dark shadow color of a 3-D element
ControlLight	The color of the light color of a 3-D element
ControlLightLight	The color of the lightest color of a 3-D element
ControlText	The color of the text in a 3-D element
Desktop	The color of the operating system desktop
GrayText	The color of dimmed (disabled) text
Highlight	The color of the background of selected items
HighlightText	The color of the text of selected items
HotTrack	The color used to designate a hot track item (hot track items are activated by a single click)
InactiveBorder	The color of an inactive window's border
InactiveCaption	The color of the background of an inactive window's title bar
InactiveCaptionText	The color of the text of an inactive window's title bar
Info	The color of the background of a ToolTip
InfoText	The color of the text of a ToolTip
Menu	The color of a menu's background
MenuText	The color of the text of a menu
Window	The color of the background of the client area of a window
WindowFrame	The color of the window frame
WindowText	The color of the text in the client area of a window

Other Supporting Classes

Two additional .NET Framework classes are of use when working with color: the ColorTranslator class and the ControlPaint class.

The ColorTranslator class does exactly what you might expect. It provides methods to translate a .NET Framework Color structure to an HTML color, an OLE color, or a Win32 color, and back, as shown in Table 4–6.

The ToHtml method is particularly useful when combined with a SystemColor. Instead of returning the color name, it returns the CSS name of the System element.

Finally, the ControlPaint class, which we examined in Chapter 2, *.NET Graphic Objects*, provides four methods for manipulating the value of a color, as shown in Table 4–7.

Table 4–6 ColorTranslator Methods

Method	Description
FromHtml	Creates a Color structure from an HTML color
FromOle	Creates a Color structure from an OLE color value
FromWin32	Creates a Color structure from a Win32 color value
ToHtml	Translates a Color structure into an HTML color string
ToOle	Translates a Color structure into an OLE color value
ToWin32	Translates a Color structure into a Win32 color value

Table 4–7 ControlPaint Color Methods

Method	Description
Dark	Returns ControlDark if the color is a SystemColor; otherwise, creates a new, darker version of the specified color
DarkDark	Returns ControlDarkDark if the color is a SystemColor; otherwise, returns a new, darker version of the specified color
Light	Returns ControlLight if the color is a SystemColor; otherwise, returns a new, lighter version of the specified color
LightLight	Returns ControlLightLight if the color is a SystemColor; otherwise, returns a new, lighter version of the specified color

These four methods can be used to easily create a set of colors for drawing three-dimensional objects, provided the base color isn't one of the SystemColors.

The Dark and Light methods optionally allow you specify a percentage value that indicates the amount by which the value is to change. Using this capability, you can create an entire suite of shaded (or tinted) hues with which to achieve some very sophisticated effects.

In the next chapter, the last in Part I, we'll examine the classes that the .NET Framework provides for manipulating images.