July, 1999

Ask Advisor

Windows Colors and Fonts

Visual FoxPro 6.0/5.0/3.0

Q: Is there any simple way to modify the text attributes for the message in the MessageBox() function? If not, is there any similar function I can use that lets me modify the attributes?

—Sam Chang (via Advisor.Com)

Q: How do I change the background color of a CommandButton? I've heard that it should follow the Windows environment default. Is that true?

—Donly Wu (via Advisor.Com)

A: I've grouped these two questions together not because both questioners are from Hong Kong (though they are), but because they raise the same issue about writing applications in the Windows environment. Both the font used for the MessageBox() function and the BackColor of buttons are controlled by Windows. More specifically, they're both settings over which the user has control through the Windows Display Properties dialog.

An easy way to open that dialog is to right-click on the desktop and choose Properties. For these particular settings, choose the Appearance page. (It's also available through the Display item in the Control Panel.) It's easy to see how the user can change the MessageBox() font. In the Item dropdown, choose Message Box and the various font items in the dialog are enabled. The user can choose the font, size, color and style (Bold or Italic) of the message in a message box.

Setting the background color of buttons in this dialog is a little more subtle. That color is controlled by the 3D Objects settings. The user can set a background color and a foreground color, which control the colors of all dialogs in Windows. A button is expected to share its background color with the dialog that contains it. (In fact, the trick used to give buttons a 3D appearance is based on the form having the same background color and isn't quite as visually effective when that's not true.)

Of course, VFP does let you change all kinds of other colors without regard for the Windows settings, so why not these? I don't really have a good answer except to turn the question around the other way. Why are we allowed to change colors and fonts without regard for the users' preferences? The answer to that one is that there are situations where something special is appropriate – for example, to get the user's attention, a particular item might need to be in an extremely large font.

However, this whole discussion points out that changing BackColor, ForeColor and the other color properties may cause problems for some users. First, many users set up their own "appearance schemes" and, in fact, Windows includes a whole collection of them (as well as so-called Desktop Themes, in some versions) for users to choose. If

your application expects the default Windows colors and makes changes accordingly, it's going to stand out for those users and not in a good way.

Far more important is that some users have visual problems of one kind or another. Both colorblindness and visual impairment mean that some users set their machines up with very specific color and/or font combinations that enable them to work. Setting custom colors or fonts in your applications is a risky business.

So how can we handle this issue in a way that respects the user, but allows us to emphasize what we think is important? The answer has two parts. First, wherever possible, use the Windows settings. By default, new forms created in VFP 5.0 and later use the Windows 3D settings. This is controlled by the ColorSource property. The default value is 4-Windows Control Panel (3D Colors). ColorSource also provides the option of using the Windows Window Colors (5); those are the ones that control documents. Whenever possible, use one of these two settings and do not directly set any of the color properties.

If you do need to specify special colors in some situations, your best bet is to find out what colors the user's machine is currently set for and then set other colors based on those. In fact, you can choose other user color settings that can be assumed to contrast well with the color settings. For example, the setting for menu text can be assumed to contrast with the setting for the menu itself. If the user has failed to make them contrast, he's likely to notice it in every application he uses, not just yours.

The GetSysColor() API function tells you what the system colors are. To use it, you first declare it:

```
DECLARE Integer GetSysColor ;
   IN Win32API ;
   INTEGER nElement
```

Once you've done that, you can inquire about the color of any individual element. You pass the number that identifies the elements. Finding out what those numbers are is actually the hardest part. The documentation for the function (which I found online in the MSDN library) uses a set of named constants. Here are the constants you're most likely to be interested in:

```
#DEFINE COLOR_SCROLLBAR            0
#DEFINE COLOR_BACKGROUND           1
#DEFINE COLOR_ACTIVECAPTION        2
#DEFINE COLOR_INACTIVECAPTION      3
#DEFINE COLOR_MENU                 4
#DEFINE COLOR_WINDOW               5
#DEFINE COLOR_WINDOWFRAME          6
#DEFINE COLOR_MENUTEXT             7
#DEFINE COLOR_WINDOWTEXT           8
#DEFINE COLOR_CAPTIONTEXT          9
#DEFINE COLOR_ACTIVEBORDER        10
#DEFINE COLOR_INACTIVEBORDER      11
#DEFINE COLOR_APPWORKSPACE        12
#DEFINE COLOR_HIGHLIGHT           13
#DEFINE COLOR_HIGHLIGHTTEXT       14
#DEFINE COLOR_BTNFACE             15
#DEFINE COLOR_BTNSHADOW           16
```

```
#DEFINE COLOR_GRAYTEXT            17
#DEFINE COLOR_BTNTEXT             18
#DEFINE COLOR_INACTIVECAPTIONTEXT 19
#DEFINE COLOR_BTNHIGHLIGHT        20
```

So, to determine the current background color for buttons, you can call:

```
nButtonBack = GetSysColor(COLOR_BTNFACE)
```

To take colors apart into their red, green about blue values, use the RGBComp() function in the FoxTools library:

```
SET LIBRARY TO FoxTools ADDITIVE
STORE 0 TO nRed, nGreen, nBlue
RGBComp(nButtonBack, @nRed, @nGreen, @nBlue)
```

Handling font settings is trickier. Although there are lots of font and size settings (title bars, menus, message boxes, etc.) available in the Appearance dialog, Windows doesn't provide a way for users to set the default font and size for either dialogs or documents. Perhaps that's because the needs of each application are so different. Your best bet here is to respect the user's settings for message boxes and other interface elements and allow the user to set fonts and sizes within your application to make it accessible to the broadest range of users.

Finally, to be sure your application isn't making unreasonable assumptions, it's a good idea to test it using several different color schemes. In 16-bit Windows, the Hot Dog Stand scheme was always a good choice for pointing out color problems. That one's gone in 32-bit Windows, but it's not too hard to create your own unusual scheme. In Windows 98, try the Baseball theme – it's pretty loud.

-- Tamar