Windows 95 Issues for FoxPro

Doug Hennig

This month's column examines issues FoxPro developers must take into account when developing applications that will run under Windows 95. We'll also look at techniques for creating cross-platform (Windows 3.1 and Windows 95) applications.

Although Windows 95 hasn't taken over the desktop by storm yet, it will. While some users are putting off upgrading their Windows 3.1 system, it's getting more and more difficult to purchase a new computer without Windows 95 pre-installed. So even if your clients don't plan to do a full migration for a while, you still need to be prepared to handle the new systems they purchase and the power users who upgrade now. In this month's "Best Practices", we look at how new features in Windows 95 affect FoxPro applications, and look at some strategies for creating cross-platform (Window 3.1 and Windows 95) objects and code.

Although there are many new features in Windows 95, the things we'll look at in this column regard user interface and long file name issues. In a future column, we'll look at other Windows 95 issues, such as using the Registry to hold application configuration information.

Detecting Windows 95

The first thing to be concerned with is how to tell when an application is running under Windows 95. The OS(1) function can be used to determine which version of Windows is running. There are a few complications, though:

- The "1" parameter for OS() isn't supported in FoxPro for DOS and Unix, so a cross-platform routine that determines if Windows 95 is running should return .F. immediately for these two products.
- In FoxPro for Windows, OS(1) returns "Windows 3.95", while in VFP, it returns "Windows 4.00". Your code must handle this difference, and be prepared for possibly slightly different values when newer versions of Windows 95 (Windows 96?) are released.

The source code disk includes ISWIN95.PRG, a very simple routine that returns .T. if the code is running under Windows 95. Here is the code

```
return _WINDOWS and ('4.' $ os(1) or '3.9' $ os(1))
```

User Interface

As you've no doubt noticed, Windows 95 has a considerably different user interface than Windows 3.1. Microsoft did a lot of research in creating the Windows 95 UI. We might as well take advantage of their efforts and follow the standards they've created. As early Macintosh developers found out, you break these standards at your peril; applications that don't follow the UI standards seem weird to many users, and are less likely to be accepted and used. Although several books have been written on the Windows 95 UI, one I've looked at is *The Windows Interface Guidelines for Software Design*, available on the Microsoft Developer Network (MSDN) CD-ROM (look under Specifications/Windows Guidelines).

Some of the new UI features are out of the control of FoxPro developers. For example, window captions are centered in Windows 3.1 while they're left-justified in Windows 95. We don't have the ability to change this, so don't worry about it. Here are a few UI things that are under our control in FoxPro.

Minimizing Bold-Faced Text

You've probably noticed that while bold-faced text was ubiquitous in Windows 9.1, it's used relatively sparingly in Windows 95. If you know your application will only run under Windows 95, you could simply use normal fonts for most objects. However, if you want your application to adapt itself at run-time (bold for Windows 3.1, normal for Windows 95), you have to use a few tricks.

In VFP, you have a couple of choices. The Init event of a form could use the SetAll method to set the FontBold property of its controls based on whether the application is running under Windows 95 or not. For example:

```
This.SetAll('FontBold', IsWin95())
```

Another possibility is to have the Init event of each control determines its own FontBold property. Since it's a good idea to subclass all the VFP base classes and use those subclasses for your controls, this could be just one more thing your subclasses do. The source code disk includes a class library called CONTROLS.VCX that contains three classes demonstrating this:

Class	Based On	Properties and Methods Added or Changed
SFForm	Form	BackColor: 192,192,192 ShowTips: .T. custom lWin95 property added Load event: This.lWin95 = IsWin95() && .T. if running under Windows 95 Init event: This.AutoCenter = .T. && centers the form at run time
SFTextBox	TextBox	FontBold: .F. FontSize: 8 Height: 21 Init event: This.FontBold = iif(type('Thisform.lWin95') = 'U', ; IsWin95(), Thisform.lWin95) && sets FontBold to .F. for Windows 95
SFLabel	Label	BackStyle: Transparent AutoSize: .T. FontBold: .F. FontSize: 8 Height: 16 Init: This.FontBold = iif(type('Thisform.lWin95') = 'U', ; IsWin95(), Thisform.lWin95) && sets FontBold to .F. for Windows 95

The Load event of SFForm initializes a custom property called lWin95 to the value returned by IsWin95(); this allows every control on the form to look at the form's lWin95 property rather than calling this routine over and over. The Init event of SFTextBox and SFLabel set their FontBold properties to .F. if the application is running under Windows 95 (they call IsWin95() if the form they're sitting on doesn't have an lWin95 property). The other property or method changes for these classes are just the way I like to set up my classes.

If you want to test these classes, create a form based on SFForm, drop an SFTextBox and SFLabel on the form, and run it. If you don't have access to a Windows 3.1 machine, simulate one by changing IsWin95() to always return .F. so you can see how this affects the appearance of the controls.

Menus

Considering that Microsoft Office has over 80% of the "office suite" market, following the Microsoft Office standards for menu layout is probably a good idea. This means having File, Edit, View (possibly), Tools, Window, and Help pads, New, Open, Close, Page Setup, Print, and Exit functions in the File menu, etc. Some of the typical Office functions may not map as easily to a database application as they do to a "document-centric" application like Word; should File, New or Insert, Record be used to create a new record, for example? However, one thing to ensure is that you use the same shortcuts for common tasks as Office applications do: Ctrl-P for Print, Ctrl-S for Save, Ctrl-N for New, etc.

Context Menus

One of the first things Windows 95 users learn to appreciate are "context" menus, those little popup menus that appear when you click the right mouse button. Context menus don't provide additional functionality; they simply allow a user to chose from a limited set of options for the current context. This is usually much easier and faster than wading through a set of menus trying to find the function you're looking for.

Context menus can help make a VFP application seem more professional. For example, if the user right-mouse clicks on a form, you might present a list of choices such as Add, Edit, and Delete. The context menu for a grid might allow the user to chose a sort order for the records from a list of tags for the table.

Context menus can be defined using the DEFINE POPUP command. The only tricky part about bringing up a context menu is placing it right where the mouse pointer is. I've found the best way to do this is to define the popup using the IN SCREEN clause and activate the menu at MROW(''), MCOL(''). We'll see an example of some code in a moment.

To make creating context menus simple, I created a class called SFPopMenu (included in CONTROLS.VCX on the source code disk). SFPopMenu provides two methods and two properties to a program or form using it:

- AddItem adds a bar to the popup menu. Pass the prompt for the bar as the first parameter and optionally a string that can be evaluated as the Skip For condition for the bar.
- Show displays the popup and sets the cPrompt and nBar properties based on the bar the user chose.

Here's the code for this class as output by the Class Browser:

```
DEFINE CLASS SFPopMenu AS custom
  cprompt = ""
  nbar = 0
  PROTECTED cpopupname
  cpopupname = ""
  PROTECTED nbarcount
  nbarcount = 0
  Name = "sfpopmenu"

  PROCEDURE show
        if This.nBarCount > 0
                activate popup (This.cPopupName) ;
                        at mrow(''), mcol('')
                This.cPrompt = prompt()
                This.nBar    = bar()
        else
                wait window 'No bars defined for popup menu.'
        endif This.nBarCount > 0
  ENDPROC

  PROCEDURE additem
        lparameters tcPrompt, ;
                tcSkipFor
        This.nBarCount = This.nBarCount + 1
        if type('tcPrompt') <> 'C' or empty(tcPrompt)
                wait window 'AddItem requires a prompt.'
                return .F.
        endif type('tcPrompt') <> 'C' ...
        if type('tcSkipFor') = 'C' and ;
                not empty(tcSkipFor)
                define bar This.nBarCount of (This.cPopupName) ;
                        prompt tcPrompt skip for &tcSkipFor
        else
                define bar This.nBarCount of (This.cPopupName) ;
                        prompt tcPrompt
        endif type('tcSkipFor') = 'C' ...
  ENDPROC

  PROCEDURE Destroy
        release popup (This.cPopupName)
  ENDPROC

  PROCEDURE Init
        This.cPopupName = sys(2015)
        define popup (This.cPopupName) margin relative ;
                shadow color scheme 4 in screen
        on selection popup (This.cPopupName) ;
                deactivate popup
        This.nBarCount = 0
  ENDPROC
ENDDEFINE
```

To instantiate SFPopMenu in a form, do the following:

- Drag SFPopMenu from the Project Manager to the form.

- In the Init event of the form, use the AddItem method of the SFPopMenu object to add bars to the popup menu.
- In the RightMouse event of the form or an object, use the SFPopMenu.Show method to display the popup and then perform some action (usually in a CASE statement) based on the cPrompt or nBar property of SFPopMenu after the user has chosen an item.

The TESTPOP.SCX form on the source code disk demos the SFPopMenu class. If you don't receive the source code disk, create a form, drop an SFPopMenu object on it, and enter the following code in the appropriate methods:

Init event:

```
with This.SFPopMenu
  .AddItem('Bar 1 - Enabled')
  .AddItem('Bar 2 - Disabled', '.T.')
  .AddItem('\-')
  .AddItem('Bar 4 - Enabled')
endwith
```

RightMouse event:

```
with This.SFPopMenu
  .Show()
  wait window 'You chose bar #' + ;
       ltrim(str(.nBar)) + ', prompt ' + .cPrompt
endwith
```

File Dialogs

FoxPro and VFP include three functions useful for allowing the user to select a file: LOCFILE(), PUTFILE(), and GETFILE(). These function really just call the equivalent native operating system functions. This has an unexpected consequence: the maximum size of the caption that can be displayed to the user is much smaller in Windows 95 than it is in Windows 3.1. For example, try the following code in Windows 95:

```
? getfile('', 'My File Name')
```

Notice the prompt in the dialog reads "My File". If an entire word cannot be displayed, it's dropped. You can see this if you shorten the prompt; anything longer than "My File Na" displays as "My File". This drastically limits the amount of text you can provide to the user explaining what you expect them to select in this dialog. "Select File to Import" or even "Select Report" will appear as "Select", which isn't very informative.

Since we don't want to penalize people who use our applications under Windows 3.1, we need to provide a more descriptive message for them while keeping the message short so it isn't cut off in Windows 95. The code shown below (included on the source code disk in LOCATE.PRG) accepts the name of a file as a parameter and displays a GETFILE() dialog asking the user to locate the file. The prompt for the user is just the name of the file in Windows 95 but "Locate" and the name of the file in Windows 3.1.

```
lparameters tcFile
local lcExt, ;
  lcPrompt, ;
  lcFile

#define ccMSG_LOCATE       'Locate <Insert>:'
#define ccMSG_LOCATE_WIN95 '<Insert>:'
#define ccMSG_INSERT       '<Insert>'

lcExt = substr(tcFile, rat('.', tcFile) + 1)
if IsWin95()
  lcPrompt = strtran(ccMSG_LOCATE_WIN95, ;
       ccMSG_INSERT, tcFile)
```

```
else
  lcPrompt = strtran(ccMSG_LOCATE, ;
        ccMSG_INSERT, tcFile)
endif IsWin95()
lcFile = getfile(lcExt, lcPrompt)
return lcFile
```

By the way, I discovered the use of STRTRAN() to substitute a variable string into a message, an idea I absolutely love and use everywhere, by reading through the code of the late Tom Rettig's public domain EDC. If you're looking for cool, elegant ideas, it's worth spending some time going over this code, which is available on CompuServe.

Other UI Goodies

VFP provides the ability to define tool tips for objects (the ToolTipText property) and information that appears in the status bar when a control receives focus (the StatusBarText property). By all means, use these tools! If you're like me, once you get used to using tool tips for infrequently used options, you get annoyed by applications that don't use them. Remember to set the ShowTips property of the form to .T. (notice the SFForm class does this), since this property is .F. (meaning tool tips aren't displayed for any object in the form) by default.

Long File Names

Windows 95 permits files to be freed of the shackles of the "8.3" naming convention. File names can even include spaces. This has some serious implications in FoxPro applications.

If your application has complete control over file names, stick with the 8.3 naming convention. However, if you allow the user to specify file names anywhere in the application (for example, for data importing, user-defined reports, etc.), you have to be prepared to handle names that are not only longer than they are in Windows 3.1 but can also contain different combinations of characters. Here are things to consider:

- If you store file names in a table, make the field long enough (or use a Memo field) to store a long name. This is especially true if the path is stored as part of the name.
- Depending on how you process a file name, you may need to handle file names in UNC format, such as \\MyServer\Directory\This is my long filename. In other words, you can't assume the first two characters of a file name are the drive letter and a colon.
- A file name can have multiple periods, which will break code like the following that separates the extension from the rest of the file name:

  ```
  lcExtension = substr(lcName, at('.', lcName) + 1)
  ```

  Instead, use the RAT() function to find the last period.
- Under some conditions, you'll need to put quotes around file names if they contain spaces. For example, while the following code will work:

  ```
  x = 'MY LONG FILE'
  use (x)
  ```

  this will give an error:

  ```
  use &x
  ```

  In this case, you need to include quotes around the name:

  ```
  x = ['MY LONG FILE']
  use (x)
  ```

- Although it's legal in VFP, avoid using spaces in table names. As well-known FoxPro developer Andy Neil says, "arsenic is faster". One of the complications of using spaces in table names is that

VFP automatically substitutes spaces with underscores in the alias. To see an example of this, create a table called MY LONG FILE.DBF, then execute the following code:

```
x = 'MY LONG FILE'
use (x)
* This will display MY_LONG_FILE
? alias()
select 0
* This gives an error
select (x)
* This works
select (strtran(x, ' ', '_'))
```

- The Windows 95 standard is to hide file extensions from users. Keep this in mind if you store file names somewhere and allow the user to select one from a list.

Conclusion

Windows 95 throws some wrinkles at FoxPro developers, especially if you're trying to develop applications that run under both it and Windows 3.1. Fortunately, there are not too many problems that can't easily be overcome.

Next month, we'll look at a development logging system we use at Stonefield to record what changes were made when to what modules in what applications. This little tool has saved us on numerous occasions and also serves as the basis of an automated system of delivering updates to people who purchase our tools.

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Sask., Canada. He is the author of Stonefield's add-on tools for FoxPro developers, including Stonefield Data Dictionary for FoxPro 2.x and Stonefield Database Toolkit for Visual FoxPro. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at user groups and regional conferences all over North America. CompuServe 75156,2326.