December, 1995

Advisor Answers

Q: Having used Visual Basic for a short time (in preparation for the release of VFP) I became spoiled by the ease of using toolbox controls such as DirListBox, DriveListBox and FileListBox for selecting directories, drives and files, respectively. I have seen a demo of VFP and was disappointed to see that these three control features are not included in VFP, and that the toolbox itself contains fewer control items. I would appreciate it if you could give me any recommendations on how to implement the directory, drive and file list boxes in my applications by building them onto my forms.

–Dean H. Neal (via CompuServe)

A: I haven't used Visual Basic, so I'm not familiar with the specific items you mention but I can guess pretty much what they do. FoxPro has a couple of options for getting at this information.

First, the GetFile(), LocFile() and PutFile()functions provide direct access to the File-Open and File-Save dialogs. In many situations, this is just what you need. (Watch out for GetFile() and LocFile(), though, in Windows - they let you return the name of a non-existent file - check it with FILE().)

If you want to integrate this sort of thing into a form (or a screen, in 2.x), you can use some of the unusual forms of list boxes and combo boxes.

Let's start with 2.x. There, you can base a list box on what's called a "Popup Prompt Files." (In FoxPro/Windows and FoxPro/Mac, the option is labelled "From Files" in the list dialog.) This gives you a mechanism for navigating directories and files. You can specify a file skeleton so only appropriate files appear.

In Visual FoxPro, a similar mechanism lets you include the same kind of navigation device in both combos and lists. Set RowSourceType to 7 and set RowSource to the file skeleton you want.

The flaw in both cases is that you can't restrict the user to a single directory. If you're interested only in directories, this doesn't work for you either. Fortunately, it's easy to create a list or combo that lists only files or only directories.

The key is the ADIR() function that lets you create an array containing information about files or directories. Use ADIR() to create the list you want, then base your list or combo on the array. The optional third parameter to ADIR() lets you specify that only directories should be returned.

In FoxPro 2.x, for a list showing files in the current directory, you'd put code something like this in the screen Setup:

```
=ADIR(aFiles,"*.*") && narrow it down if you want
```

Then, the list should specify From Array aFiles. For a list of subdirectories of the current directory, use:

```
=ADIR(aDirs,"","D")
```

and base the list on aDirs. You'll probably want to parse out the first two entries in the list because they reference the current and parent directories (. and ..). You can do:

```
=ADEL(aDirs,1)
=ADEL(aDirs,1)      && Row 1 again because previous row 1 was
             && deleted
IF ALEN(aDirs,1)>2
   DIMENSION aDirs[ALEN(aDirs,1)-2,ALEN(aDirs,2)]
ELSE
   DIMENSIOn aDirs[1,1]
   aDirs[1,1]="No directories"
ENDIF
```

In Visual FoxPro, it's easy to create some classes that do the whole job. We can start by subclassing the ListBox base class. In this subclass, which we'll call ArrayListBox, set RowSourceType to 5 (Array). Now add an array property to the class, so it can always hold its own data. Use the New Property option on the Class menu and create a property aListData[1] - remember you need the "[1]" to tell VFP that it's an array. Set RowSource for the list to THIS.aListData and set ColumnCount to 1. Finally, resize the list to something that looks about right to you. Save this class.

Now, we'll create two subclasses of ArrayListBox, one for files and for directories. In each case, we'll use a method to populate the array and update the display. We'll also set each one up to populate and update when the object is initialized (the Init method) and when it's refreshed (the Refresh method). (We could instead add an empty method to ArrayListBox and then redefine it in each of our subclasses. The advantage of doing so is that more is done at a higher level in the class hierarchy. The disadvantage is that the method name doesn't fully reflect what it does.)

The file version is a little simpler so let's start there. We'll call this class FileListBox and it's based on the ArrayListBox class we already created. I'm putting all three classes in a visual class library called Tools, so I issue this command to get things started:

```
CREATE CLASS FileListBox OF Tools AS ArrayListBox FROM Tools
```

We don't need to change any of the properties. The work we need to do is in methods. First, we add a new method to the class called GetFiles. Mark the method as protected so it can't be used outside the class itself. Here's the code:

```
IF ADIR(THIS.aListData,"*.*") = 0
   DIMENSION THIS.aListData[1]
   THIS.aListData[1] = "No files"
ENDIF
THIS.ReQuery()
```

We fill the array property aListData with the list of all files in the current directory. Then, we make sure there are some. If not, just create one item containing "No files." The last line re-populates the list with the updated array data.

The rest is easy. We just need to call our GetFiles method from two other methods. In the Init and Refresh methods, put:

```
THIS.GetFiles()
```

Now save the class and we're done. You have a listbox that displays all the files in the current directory. When you change directories, just call the list's Refresh method and it's updated.

On to directories. This one's a little more complicated because we need to remove the entries for the current and parent directory, but the principle is the same. Add a custom, protected method GetDirs and put this code in it:

```
LOCAL nRows
nRows=ADIR(THIS.aListData,"","D")
=ADEL(THIS.aListData,1)
=ADEL(THIS.aListData,1)
IF nRows > 2
   DIMENSION THIS.aListData[nRows-2,ALEN(THIS.aListData,2)]
ELSE
   DIMENSION THIS.aListData[1,1]
   THIS.aListData[1,1]="No subdirectories"
ENDIF

THIS.ReQuery()
```

Then put `THIS.GetDirs()` in the Init and Refresh methods. Save the class and you're done.

You can drop any of the three list boxes into any form and they'll work. (If you use ArrayListBox, you'll need to add code to fill the array.) Figure 1 shows a form with one of each type.

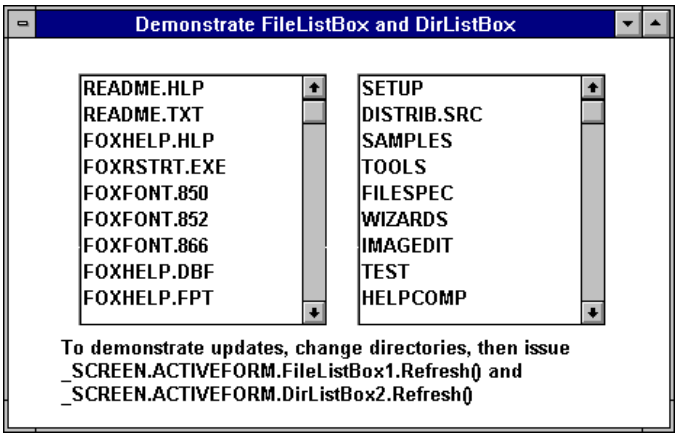

Figure 1. Once you've built the FileListBox and DirListBox classes once, you can pop them onto any screen and they'll work just as if they were built in.

I can think of some enhancements we might want in these classes. One is to be able to pass a file mask to FileListBox and only see files that match that spec. We can do that by subclassing FileListBox to accept the necessary parameter. The new subclass would need another custom property to hold the mask. Then, a new version of GetFiles would need to use the input mask property in its call to ADIR(). I'll leave the actual details as an exercise.

Other changes might include displaying the information in mixed, rather than upper, case or sorting the lists. Again, you can accomplish this by subclassing these classes.

You'll find TOOLS.VCX containing ArrayListBox, FileListBox and DirListBox and the sample form, TestList.SCX on this month's Companion Resource Disk.

Getting your hands on file and directories is pretty simple. Unfortunately, there's no native way to do the same for drives. I suspect there's a Windows API call that would provide this information - Ted (who knows more about these things than I do) suggests using either the Windows Api or FoxTool's DriveType() function in an A through Z loop. Then a similar class can be developed.

–Tamar