The Fox Hunt

Switching between development and user mode... -> EXE in root, IT.PRG in source dir

By Whil Hentzen

Most programmers start by placing all the files of their application in a single directory. It's easy to program, since you don't have to worry about complicated pathing and directory structure issues, and it's easy to find everything. However, as the level of sophistication of your applications grows, you'll find an increasing number of limitations. For example, as the number of files grows, the performance of your application will suffer as DOS spends more and more time searching for specific files. You may also find it more and more difficult to properly provide updates - it's all to easy to accidentally overwrite a large data file of theirs with a test data file of yours when installing new programs. And as the number of programs grows, you may encounter difficulties in version control - which files have been updated and which have not?

One solution is to separate the files of your application into a number of directories according to the type of file. Let's take a look at what these file types are. First, you've got data files - and in many cases, you've likely got multiple copies of data. The second type are the applications data dictionary files. The third is source code, and the fourth is common programs, libraries, and third party utilities - files that are used across applications.

Consider breaking up your application into four directories, one for each of these types of files. Here's why. Once you've delivered an application, and the user has started entering real data, you likely wouldn't want to supply a new set of data files. Furthermore, you may want to provide the ability for the user to switch between multiple sets of data, and keeping the data separate from the rest of the system enables the application to be "pointed" toward one data set or another. You probably won't update the data dictionary files either. However, since the same data dictionary would contain descriptions of each set of data files, you'd want to keep it in a separate directory that is referred to regardless of which data set is being used.

Source code, on the other hand, will likely be updated on a regular basis, and keeping it in it's own directory will make it easy to update just those files without touching the data sets or data dictionary. However, I've found that I rarely update common code, libraries or third party tools, and since those files can take up several megabytes of space, I place those in a separate directory so I don't have to worry about zipping up the appropriate files.

Now that we've figured out how to separate our files, how do we get the application to recognize all of these files? The main application file, be it an .APP or an .EXE, resides in the directory above each of these, and looks for a system file that contains the paths of the directories in question. I used to store just the relative path but now with Macintosh file and path naming conventions, I've found it safer to store the entire path, including drive letter.

Another tip: I keep separate entries in the system file for the location of these directories in my development environment and in the customer or user's environment. I may be doing development on drive F while they have it installed on drive T. Nothing is more frustrating than bringing out a new copy of the app with a new system table, only to have the application crash because it's looking for paths that don't exist. I use a DOS environment variable to tell the system which set of directory names to look for. For instance, the system table, SYS_SYST.DBF, might contains entries for the common directory for the developer and for the user:

[This should be a screen shot of a table with several entries...]

cSysDesc	cSysData
DeveloperCommonLocation	F:\COMMON26

```
UserCommonLocation      T:\FOXPRO\COMMON26
DeveloperDDLocation     F:\DEALER\DDFILES
UserDDLocation          T:\FOXPRO\DEALER\DDFILES
```

If the application detects a DOS environment variable named “fpdev” and it’s set to “WHIL”, then the application will use the paths for my machine, otherwise, it will use the paths for the user’s machine.

```
select SYS_SYST
if upper(getenv(“fpdev”)) = “WHIL”
 locate for cSysDesc = “DeveloperCommonLocation”
else
 locate for cSysDesc = “UserCommonLocation”
endif
m.gcCommLoc = cSysData
if upper(getenv(“fpdev”)) = “WHIL”
 locate for cSysDesc = “DeveloperDDLocation”
else
 locate for cSysDesc = “UserDDLocation”
endif
m.gcDDLoc = cSysData
```

Now the application can include m.gcCommLoc and m.gcDDLoc in the path while setting up the application so that those files can be found while the application is running:

```
m.gcPath = set(“PATH”) + “,” + m.gcCommLoc + “, “ + m.gcDDLoc
set path to &gcPath
```

Setting up this architecture is a bit of work, but once it’s in place, it can be reused from application to application, and the formerly arduous process of keeping files straight between your site and the users’ becomes a no-brainer.

Next month, we’ll address the issue of being able to transparently run your application both from a compiled .APP or .EXE or from the original source code, even though they are kept in different locations.