The Fox Hunt

By Whil Hentzen

One of the nice capabilities of FoxPro is the ability to compile an application into a single file and distribute it - either as a standalone EXE, an EXE with run time files, or an .APP that runs with FoxPro or from a parent EXE file. While you can place this compiled file in the same directory that contains the source code, most experienced developers do not.

I place the .APP or .EXE directly above the source code directory for a number of reasons. First, when shipping the application, the resulting directory structure is clean. The compiled file and a few others reside in the root, and all other files - such as the data dictionary files and data itself - are safely tucked away in their own directories. Second, it's likely that the executable is not the only file that needs to be shipped with the application. If you compile and run the app from the source code directory, you're relying on the supporting files in the directory - and it's all too easy to forget to include one of them when sending the final app to the customer. And finally, if you're in the habit of naming the project and main program with the same name, it's easy to compile an APP with the same name as the main PRG, make some changes in the source code, and then wonder why the changes didn't take effect. (If you don't recompile the APP, the next time you run the .PRG - so you think - you're still running the old .APP. This bites everybody at one point or another.)

However, there's one problem. The files that your application needs to find are in different relative locations depending if you're running from the source code directory or from the .APP file "above" the source code. Even if you're using a "SYS_SYST.DBF" type of system file that stores the location of your data sets and common code (as we discussed last month), you still need to find this file. Keeping two copies of this file - one in your source code directory and one with the compiled file - it's just too easy to make a change to one and not the other. Of course, this isn't a problem if you always build the .APP while testing, but many developers don't. I always run from the source code directory and build the .APP as the last step in my development process.

The solution to this problem is rather simple - in your setup code, examine where the program is being started from, and then look for the SYS_SYST.DBF file (I keep mine in my data dictionary directory) based on that answer. As long as you always keep the data dictionary files "next to" the source code and "under" the compiled .APP or .EXE, you can successfully look for the files regardless if you're running from the source code directory or from the .APP itself.

First, we determine the location of the startup directory. If we're running from the source code directory, the last few characters will be "SOURCE" and we can assume that our data dictionary files (including SYS_SYST.DBF) will be in a directory on the same level. Note that we still make sure that SYS_SYST.DBF exists. The m.glWeAreDone will bail out of our program if the file doesn't exist.

```
*
* save the Directory that was the default at the start of this program
* it's either something like
*  "f:\borg\source" (for developing) or
*  "f:\borg" (for production)
*
m.gcDefAtProgStart = sys(5) + sys(2003) && Default at Program Startup

sele 0
if uppe(right(m.gcDefAtProgStart,6)) = "SOURCE"
 * assuming APPFILES is "next door"
```

```
 if file(left(m.gcDefAtProgStart, len(m.gcDefAtProgStart) - 6) + "APPFILES\SYS_SYST.DBF")
  use left(m.gcDefAtProgStart, len(m.gcDefAtProgStart) - 6) + "APPFILES\SYS_SYST"
 else
  wait wind left(m.gcDefAtProgStart, len(m.gcDefAtProgStart) - 6) + "APPFILES\SYS_SYST.DBF" + " does
not exist"
  m.glWeAreDone = .T.
 endi
else
 * assuming APPFILES is "down below"
 if file(m.gcDefAtProgStart + "\APPFILES\SYS_SYST.DBF")
  use m.gcDefAtProgStart + "\APPFILES\SYS_SYST"
 else
  wait wind m.gcDefAtProgStart + "\APPFILES\SYS_SYST.DBF" + " does not exist"
  m.glWeAreDone = .T.
 endi
endi
```

Once we've opened up SYS_SYST, we look for the paths that point toward our common and application files.

```
*
* in either case, we have opened SYS_SYST.DBF
*
if !m.glWeAreDone

 if upper(getenv( "fpdev" )) = "WHIL"
  locate for cSysDesc = "DevCommonLocation"
 else
  locate for cSysDesc = "UserCommonLocation"
 endif
 if !eof()
  m.gcCommLoc     = allt( SYS_SYST.cSysData )
 else
  m.gcCommLoc = ""
  wait wind "There is no Common Location in SYS_SYST"
  m.glWeAreDone = .T.
 endi

 if upper(getenv( "fpdev" )) = "WHIL"
  locate for cSysDesc = "DevDDLocation"
 else
  locate for cSysDesc = "UserDDLocation"
 endif
 if !eof()
  m.gcDDLoc   = allt( SYS_SYST.cSysData )
 else
  m.gcDDLoc = ""
  wait wind "There is no DDFiles Location in SYS_SYST"
  m.glWeAreDone = .T.
 endi
```

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

endif
```

As with the initial Common, Data Dictionary, and Source Code directory structure described last month, setting up this architecture is a bit of work, but once it's in place, it can be reused from application to application. And just like those other handy tips and tricks, after you're used to using it, you'll wonder how you ever lived without it.