Easing the Testing Burden

By Whil Hentzen

An application that can't be tested won't be tested. And unless you write perfect code the first time (happens to me All The Time), an application that isn't tested is going to break - when your customer is using it.

But testing isn't any fun. In fact, it's a pain. However, you can ease the testing burden through the technique of having separate directories for test and live databases, and for development and production programs.

Here's a step by step procedure for setting up this environment.

Set up your directories

For the sake of this article, we're going to assume that all of our source code and design surfaces (menus, screens, reports) are in the same directory. Once you understand the concepts presented here, it's easy to handle the subdirectory structure of PROGRAMS, SCREENS, MENUS, REPORTS and so on.

We're going to set up a main directory for each company (or department) we work with, and then create subdirectories for each application for that company. Under every app, we'll use the same set of six directories. Here we're developing for the Widget Company, and currently have Inventory and Dealer systems:

```
WIDGET_CO
  INVENTORY
    DOC
    SOURCE
    TEST
    TEST_ORI
    LIVE
    PROD
  DEALERS
    DOC
    SOURCE
    TEST
    TEST_ORI
    LIVE
    PROD
```

DOC contains all documentation and notes for the system. I keep it on the same level as everything else for ease of backup.

SOURCE and PROD contain the source code and the production versions of our system. SOURCE contains the system's project, programs, design surfaces, system files and so on. PROD contains the .APP (or .EXE) and copies of files that belong to the system but can't be bound into the APP (such as an updatable system table).

TEST and LIVE contain data sets. The test data is generally a subset of data that the user can work with without fear. The live data is the user's actual business information. TEST_ORI contains a master copy ("TEST ORIGINAL") of the test data.

The key to this structure is that all the directories are on the same level, and we can refer to the contents of any directory from any other directory through relative addressing. In other words, we don't have to hardcode a path. If we're in the SOURCE directory and want to open up the PARTS table in TEST, we can issue the command

```
use ..\TEST\PARTS
```

instead of

```
use C:\WIDGET_CO\INVENTORY\TEST\PARTS
```

This way, we can install the production version of the system on any drive, and at any level of the directory structure.

There are more advantages to this structure that we'll see in a few minutes.

Create test and live data sets

It's beyond the scope of this article to discuss in detail the proper construction of a test suite of data. However, let me point out that a well designed test suite should help _you_ test the application (boundary conditions, null sets, and so on), but _also_ provide a road map for the user.

They'll use this test data to learn the system, practice with a feature that they haven't used in a while, and to train new users. It's important to populate this test data so that it is sized and scoped realistically in terms of their business. For example, if the user works in a machine shop, you don't want to use "surfboards" and "hot dogs" as descriptions for work in process items.

Furthermore, you'll want to use data items that are easily queried against. If they have to select a "type of part", use their jargon - screws, bolts, nuts, washers - not nonsense like table, airplane, tricycle, and Daffy Duck.

Be sure to copy the test data to the TEST and TEST_ORI directories once you're finished creating it. Nothing is more frustrating than to run the system against a well-crafted test data set, and then blow three days recreating it because you forgot to make a master copy.

Naturally, your live data set will consist of seed tables and other files that will be initally shipped with the app.

Create the system table

Our system table will need one field where we store the relative location where the user last worked - both on a day-to-day basis as well as between operations during a single session.

We're going to use the field name cMRDataLoc (a character field that contains the Most Recent DATA LOCation.) My main system file is always called IT.DBF - short to type and fun to work with ("now we're going to use it - no, use IT").???This table is in the system directory, right, not in the data directory? Clarify this. --Tamar???

??? 93/11/6 Good point. I should have made explicit that the system table - IT.DBF _and_ the system data dictionary files (usually, SYS* - XCATs stuff, and FFDAT - Foxfire! stuff) are both in the SOURCE directory. If these tables are read only, they can be bound into the .APP/.EXE for distribution. However, they are usually modifiable (for example, IT.DBF is not only where cMRDataLoc is (user modifiable), but also where the "Last Invoice Number" stuff is) so I don't bind.???

??? 93/11/6 additional wording for the above paragraph to make this clear:

Note that IT.DBF and any other system tables (like those you'd use for a data dictionary) reside in SOURCE, not in the data directories. When I compile the application and place the resulting .APP or .EXE into the PROD directory, I also include a seed copy of those tables in the PROD directory.

???

The contents of IT.cMRDataLoc is "..\TEST" or "..\LIVE".

Create the main (calling) program

When we start our system, we'll either run a main program from the SOURCE directory or run an .APP from the PROD directory. In either case, we're going to switch from the directory that contains the application program to the directory that contains the data (using SET DEFAULT).

We'll provide utilities for the user to switch from one data directory to the other (TEST to LIVE and back), and we'll return to the original directory when we're done. Note that this has to work regardless of whether the user started from PROD or we started from SOURCE.

Let's look at the section of our main program that handles this for us. (By the way, I call my main program IT.PRG as well. Having 21 years of marathoning under my belt, I like to "just DO IT.")

First, we're going to find out where the user last was:

```
 use IT in sele(0)
 m.gcCurDataLoc = IT.cMRDataLoc && "Most Recent Data Location"
```

Next, we're going to find out where we're starting from and store that. SYS(2003) is the current default directory, so

```
 m.gcDefAtProgStart = sys(2003) && Default at Program Startup
```

Next, save the FoxPro path that was in effect at the start of this program. We're going to be changing the FoxPro path shortly, and while the user probably doesn't have one, we probably do (COMMON). Therefore, we're going to want to save it so we can return it to its original status at program end. Our FoxPro path will be (or include) a reference to a common directory: "\COMMON20", "\COMMON25", and so on.

```
 m.gcPathAtProgStart = set("path") && FoxPro path at Program Startup
```

Now we're going to create a new path string that consists of the default directory at program startup and the existing path. We're going to be changing the default directory in a moment, but we want to path to the directory where all our programs are. If we don't include the default directory, we won't be able to find the program once we switch.

Note that the business with the IIF is necessary because we need a semi-colon to separate the default path from an existing FoxPro path.???I've changed this. Make sure it's still correct. -- Tamar???

??? 93/11/6 Actually, not exactly. The key here, Tamar, is that if the user had a path, m.gcCurPath will look like "\COMMON25; \WHATEVER" but if they didn't, m.gcCurPath will look like "\WHATEVER". I use the IIF to insert the semi-colon between the paths IF there already was one.

If m.gcCurPath = "; \WHATEVER" the system will puke.

So the semi-colon is to separate the default from the existing, but the IIF is to _conditonally_ insert it if needed. ???

```
 m.gcCurPath = ;
  gcDefAtProgStart + ;
  iif(!empt(m.gcPathAtProgStart), ";"+m.gcPathAtProgStart, "")
```

Now set the path to this new string.

```
 set path to &gcCurPath
```

Finally, set the default directory to the location where the user last left it - this is the location of the data set with which the user was last working.

```
 set defa to &gcCurDataLoc
```

We're now in the correct data directory but have access to programs in other directories.

Switching data sets

Now that we've set up the directory structure and environment, we want to provide the user with the ability to change from one set of data to the other. We'll put an option on the menu that will say either

Switch to Live Data

or

Switch to Test Data

We'll use the same menu bar and simply vary the prompt. The prompt is

```
 "+m.cTorLPrompt+"
```

and the code in the menu setup is

```
 do case
 case IT.cMRDataLoc = "..\TEST\"
  m.cTorLPrompt = "Switch to Live Data"
 case IT.cMRDataLoc = "..\LIVE\"
  m.cTorLPrompt = "Switch to Test Data"
 endc
```

Once they choose the menu option, we'll change the prompt and change the contents of the field cMRDataLoc, depending on the original location. After we're done with the CASE construct, the steps are the same:

(1) close the existing data files;
(2) switch to our default and open up IT.DBF again (CLOSE DATA also shuts down IT.DBF);
(3) switch to the new data directory;
(4) open up the data files in that directory.

??? 93/11/6 Breaking out the steps here looks a lot better!???

Finally, since we changed the menu, refresh its display as well (yes, I also call the main menu IT.MPR - saves on typing since it's considerably shorter than something like "MAIN"). And we'll provide a message reminding the user which directory they are now in.

The actual code looks like this:

```
 sele IT
 do case
 case IT.cMRDataLoc = "..\TEST\"
  m.cTorLPrompt = "Switch to Test Data"
  repl IT.cMRDataLoc with "..\LIVE\"
 case IT.cMRDataLoc = "..\LIVE\"
  m.cTorLPrompt = "Switch to Live Data"
  repl IT.cMRDataLoc with "..\TEST\"
 endc
 m.gcCurDataLoc = IT.cMRDataLoc
 * IT is in gcDefAtProgStart
 clos data
 set defa to &gcDefAtProgStart
 use IT
 set defa to &gcCurDataLoc
 do zFOpener      && library routine to open files
 do IT.MPR
 wait wind nowa "You are working with "+ subs(m.gcCurDataLoc,4,4) +" data"
```

Restoring the original test data

The purpose of having test files is being able to experiment with the data without worrying about irretrievably damaging data. However, since the user can change (and even completely delete) the test data, we need to provide the user the ability to restore the test data to it's original state.

The option in the menu will read:

Restore Original Test Files

and the code attached to this option is:

```
 run copy ..\test_ori\*.* ..\test\*.*
 clea
```

Cleaning up

When the user has decided to quit, we'll need to set everything back to the way it was. These two lines handle this quite nicely:

```
 set defa to &gcDefAtProgStart
 set path to &gcPathAtProgStart
```

Conclusion

Once you've committed to setting up your development environment to aid testing, you'll see a significant decrease in errors - both in those that you make yourself, and those that your customer sees. You'll also find your productivity enhanced through making your applications more consistent.

(Whil Hentzen is blah blah blah...)???Do you want to publish it just this way? <g> -- Tamar???

??? 93/11/6 Sure. <g>. Oh, let's see, what will sound pompous and high-faluting, yet warm, human, and approachable... <g> Let's just use the SOS:

(Whil Hentzen is president of Hentzenwerke, a 10 year old firm that specializes in FoxPro-based strategic database applications. He has commercial products and custom applications running throughout the U.S. and in 12 foreign countries, has spoken at FoxPro conferences in the U.S. and Canada and heads up the Milwaukee Association of FoxPro Developers. Voice: 414.332.9876, CompuServe 70651,2270.)

???