December, 1996

Editor's View

What's Old is New Again

What happens when you have to bring an old application up to snuff?

By Tamar E. Granor, Editor

I've been working on an interesting project for the last few months. The first application I ever wrote for a client was for my husband's business and was installed nearly two decades ago. The app is an accounts payable system designed to handle cost accounting just the way this company wants it. It was written in Wang Basic to run on a machine with 16 KB (yes, kilobytes) of memory and three 8" floppy drives totaling about 750 KB of storage. I wrote it the summer between college graduation and graduate school (with a number of enhancements since).

For the last ten years or so, we've been searching for a product to replace this application. We've even bought a couple of them. In one case, the company went out of business before they completed necessary modifications. The other package turned out to be horribly buggy and unusable. So, my homegrown application continues to be used.

As time has passed, we've been increasingly nervous about storing critical information in a form that requires aging hardware. The company went so far as to buy a duplicate machine to use for spare parts (and they've been needed more than once).

A few months ago, a consultant demonstrated a product that would let us run our existing code on the office network and end the company's reliance on the old Wang 2200. It's basically a translation layer that converts the instructions into an appropriate format and passes them on to the PC. The company bought the product and quickly moved current projects onto the network. (Archived data is slowly being moved onto the network, as well.)

The new environment for this old application has some interesting effects. First, the system runs much faster than it ever did. With data stored on the network drive rather than floppies, response is pretty much instantaneous. For the first time, however, the issue of multi-user access has arisen. As long as the application ran on a stand-alone machine, multi-user was a non-issue. Now we need to ensure that two users don't try to access the same files at the same time.

Moving to the network has changed some of the limits of this application, too. On the old machine, both the memory limit and the moderate disk storage meant we had to play a lot of tricks to get the job done. Now, memory and storage are essentially unlimited.

Because of limitations of the language and the hardware limits, program modifications were needed for each new venture the company undertook. Each had its own set of floppies with both code and data. This was, of course, a maintenance nightmare. With several ventures underway simultaneous, each time the program was enhanced or a bug was fixed, I had to remember to propagate the changes to every active venture.

Now that memory is not an issue, we've decided to create a single, generic version of the code that uses stored data to configure itself. So, for several months, once a week, I go on-site and work on modifying the code to create this generic version. Along the way, I've been addressing various problems that have been deferred over time because we always felt we were going to replace the application "real soon now."

Working on my own old code has been an interesting experience. Some of the code hasn't changed much since it was written; I, on the other hand, have changed a lot. Some of the code makes me cringe (though much of the ugliest code was written to work around hardware limitations) while other pieces astound me with their cleverness. There are many places where I'd love to apply the things I learned in graduate school and since, even a few places where I long to apply object-orientation (especially inheritance and subclassing). The worst piece of it is that, because even comments used up precious memory, there are very few of them.

Nonetheless, I've been chugging away making changes and even writing a few new routines (and loving some of the enhancements in this version of the language, like variables names composed of more than a letter and a digit, and IF statements that can be followed by more than a line number). The new code I'm writing is much more readable and easier to debug than the original code. And, of course, I'm adding copious comments to everything I touch.

So what does all this have to do with FoxPro? Not much, except for some of the lessons I'm learning. One lesson is to take pity on the poor fool who'll have to maintain your code—it might be you. The other big lesson is that, even though I've been focusing exclusively on FoxPro for a long time now, I certainly can write good code in other languages (even antiques). Some of the mistakes I make as I change mindsets are funny (when I'm not thinking about it, I often type MODI COMM to start editing a program), but my brain is capable of remembering the details of multiple development environments and, when it doesn't, there are always the manuals.

I wasn't enthusiastic at all when we made the transition to the network (though I *was* awed by it). What I really wanted was to find an off-the-shelf or semi-custom app to replace my old code. But, to my surprise, it's been fun and challenging working on it, cleaning it up, and bringing it up to date. I guess that's why I went into this business in the first place.