The Minimum You Need to Know

About Java and xBaseJ

Roland Hughes

Logikal Solutions

ISBN-13 978-0-9823580-3-0

Source Code License

Table of Contents

Introduction

Why This Book?

I asked myself that same question every day while I was writing it. Why am I going to write a book much like my other books and give it away for free? The simple answer is that I had to do a lot of the research anyway. If I have to do that much research, then I should put out a book. Given the narrowness of the market and the propensity for people in that market to believe all software developers work for free, the book would physically sell about two copies if I had it printed. (Less than 1/10th of 1% of all Linux users actually pay for any software or technology book they use.)

What started me down this path was a simple thing. In order to make a Web site really work, a family member needed to be able to calculate the 100 mile trucking rate for the commodity being sold. The commercial Web site had a really cool feature where it would automatically sort all bids within 300 miles based upon the per-bushel profit once the transportation costs were taken out. The person already had a printed list of the trucking rates, so how difficult could it be?

Some questions should never be asked in life. "What could go wrong?" and "How difficult could it be?" are two which fall into that category. When you ask questions like that, you tend to get answers you were unprepared to hear.

In my early DOS days, I would have sat down and begun coding up a C program using Greenleaf DataWindows and the Greenleaf Database library. Of course, back then, we didn't have the Internet, so I would have had to use the Greenleaf CommLib to dial out to some BBS to get the DOE (Department of Energy) national average fuel price.

During later DOS days, but before Microsoft wrote a task-switching GUI that sat on top of DOS and that was started by typing WIN at the C:> prompt which they had the audacity to call "The Windows Operating System," I would have reached for a C/C++ code generator like Pro-C from Vestronix (later Pro-C Corp.) or DataBoss from Kedwell Software. Neither program did communications, but both could be used to quickly lay out xBASE databases, generating entry/maintenance screens, menus, and reports in a matter of minutes. You could create an entire application that used just a few files for distribution, all of which could be copied into a single directory, and the user would be happy.

Once Windows came out, things got ugly. I did a lot of OS/2 work, even though not many companies or people used it. The problem with OS/2 was that IBM had Microsoft develop it and Microsoft was intent on ensuring that OS/2 would never be a threat to Windows. (Windows wasn't even an actual operating system until many years after OS/2 came out.) Once IBM had the bulk of the Microsoft code removed from it, OS/2 became an incredibly stable platform which managed memory well. IBM didn't manage it well, saddling developers with expensive device driver development tools that would only work with an increasingly hard-to-find version of Microsoft C.

Most of us did cross-platform development in those days. I used Watcom C/C++ for DOS, Windows, and OS/2 development (now OpenWatcom as the project is OpenSource). It was easy when you used the Zinc Application Framework for your GUI. There were a ton of cross-platform indexed file libraries. Greenleaf supported a lot of compilers and platforms with its Database library for xBASE files. Of course, there were a lot of shareware and commercial Btree type indexed file systems out there. These had the advantage of locking the user into your services. These had the disadvantage of locking the user into your services. They weren't widely supported by "common tools" like spreadsheets and word processors. The one I remember using the most was C-Index/II from Trio Systems LLC. As I recall it was written generically enough that it actually worked on DOS, MAC, Windows, and OS/2. Of course, that was during the brief period in life when the Metrowerks CodeWarrior toolset supported the MAC.

In short, from the 80s through the end of the 90s we always had some way of creating a stand-alone application with its own indexed local data storage that didn't require lots of other things to be installed. When Windows started going down the path of "needing lots of other stuff" was when you started seeing companies selling software to do nothing other than develop installation programs for Windows.

As an application developer who is quite long in the tooth, I don't want to link with DLLs, shared libraries, installed images, or any other thing which is expected to be installed on the target platform. I have heard every justification for it known to man. I was there and listened to people slam my C program because their Visual Basic (VB) application took only 8K and "looked slicker" than my application which consumed an entire floppy. I was also there to watch production come to a screeching halt when a new version of the VB run-time got installed to support some other "mission critical app" only to find all previous apps were now incompatible. (The machine running my C program which took a whole floppy continued to keep the business it supported running while much screaming and finger-pointing was going on all around it.)

Why this book? Because the person downloading your SourceForge project or other free piece of software doesn't consider recompiling a Linux Kernel a fun thing to do in his or her free time.

Why this book? Because Mom and Dad shouldn't have to take a course on MySQL administration just to enter their expenses and file their taxes.

Why this book? Because I had to do all of this research, which meant I had to take a lot of notes anyway.

Why this book? Because OpenSource libraries don't come with squat for documentation.

Why xBaseJ?

That's a good question. Part of the answer has to do with the history I provided in the previous section. The other part has to do with the language chosen.

I don't do much PC programming anymore. I needed this application to run on both Ubuntu Linux and Windows. There isn't a "good" OpenSource cross-platform GUI library out there. Most of the Linux GUI libraries require a lot of stuff to be installed on a Windows platform (usually the bulk of Cygwin) and that requires writing some kind of installation utility. Let's just say that the OpenSource installation generation tools for Windows haven't quite caught up to their expensive commercial counterparts. You don't really want to saddle a Windows machine which has the bare minimum Windows configuration with something like Cygwin on top of it.

When I did do PC programming, I never really did much with TCP/IP calls directly. If I magically found an OpenSource cross-platform GUI which did everything I needed on both Linux and Windows, I was still going to have to find a cross-platform TCP/IP library. Let us not forget that some 64-bit Linux distros won't run 32-bit software and some 32-bit software won't run on 64-bit versions of Windows Vista. Programming this in C/C++ was going to require a lot more effort than I wanted to put into something I would basically hand out free once it was working correctly. (You may also have noticed I didn't even mention finding a library which would work on Windows, Linux, *and* MAC.)

Java, while not my favorite language, tends to be installed on any machine which connects to the Internet. Most Windows users know where to go to download and install the JRE which isn't installed by default for some versions of Windows. From what I hear, the pissing contest is still going on between what is left of Bill Gates's Evil Empire and what is left of Sun.

Java, while not my favorite language, provides a GUI for almost every platform it runs on.

Java, while not my favorite language, makes opening a URL and parsing through the text to find certain tokens pretty simple if you happen to know what class to use.

Java, while not my favorite language, will not care if the underlying operating system is 32- or 64-bit.

Most machines which use a browser and connect to the Web have some form of the Java Run-time Environment (JRE) installed on them. This is true of current Linux, MAC, and Windows machines.

Obviously I was going to have to develop this package with a language that wasn't my favorite.

The only question remaining was data storage. Would I force Mom, Dad, and Aunt Carol to enroll in a MySQL administration course even though they can barely answer email and find MapQuest on the Internet, or was I going to use something self-contained? Given my earlier tirade, you know I wanted to use something self-contained just to preserve my own sanity.

At the time of this writing, a search on SourceForge using "java index file" yields just shy of 29,000 projects and a search using "java xbase" yields just shy of 20,000 projects. Granted, after you get several pages into the search results, the percentage of relevancy drops exponentially, but there are still a lot of choices. Btree type indexed files which store the index in the file with the data tend to be far more reliable from a data integrity standpoint. All indexes are always kept in sync by the library/engine. xBASE type files store the indexes off in different files. You can add all of the records you want to an xBASE data file without ever updating an index.

I can hear the head-scratching now. "But if that's true, why would you use 30-year-old xBASE technology instead of a Btree?" Because of the tools, child. OpenOffice can open a DBF file in a spreadsheet to let a user view the data. If any of these files become corrupted there are literally hundreds of xBASE repair tools out there. If a user decides he or she wants to load the data into some other database format for analysis, there are tools out there to export xBASE into CSV (Comma Separated Value) files which can easily be imported by most relational database engines. Some relational database engines can directly import xBASE files. Nearly every programming language out there has some form of xBASE library, or can call one written in some other language. Perl even has an xBASE library that I've used to extract data from an expense tracking system before. Under Linux, there is even a dbf_dump utility (dbfdump on OpenSuSE for some reason) which will let you dump a DBF file to a CSV in one command.

```
dbfdump /windows/D/net_f/xpns_$tax_year/payee.dbf > payee.csv
```

What happens if I use one of those really fast Btree or B+tree libraries and the user needs to get the data out? Such users cuss me pretty hard when none of the office suites on their computer can open the file to do an export. When they track me down via the Web and call my office, they get disappointed finding out I don't have time to drop everything and fly to their location to help them free of charge. Then they say my name, spit, and start bad-mouthing me all over the Internet. *That* really helps my consulting business.

Now that we have determined the data will be stored in an xBASE file format, we only have to choose an OpenSource xBASE library for Java. I selected xBaseJ because it used to be a commercial library known as XbaseJ and was sold by BMT Micro. The product has since become an OpenSource Project which gets periodic improvements. The developer actually monitors his SourceForge support forum and seems to be actively adding new things to the library. Some things don't work out so well, like the DTD to xBASE XML parser, but the attempt was made. Someone else in the community might finish it.

Please pay attention to the thought process of this section. A seasoned systems analyst and/or consultant goes through exactly this thought process when he or she tries to design a system. You look at what is available on the target platform, then walk backwards trying to reduce the amount of pain you feel when you can't change the target platform. I cannot change the computers people have, nor their personal skill levels. I have to design an application based upon the ability of the user, not my ability to be creative, or the tools I would prefer to use.

A Brief History of xBASE

There are many variations in the capitalization of xBASE, which is I guess fitting, since there are many slight variations for the actual file formats. The history of xBASE is a sordid tale, but all versions of xBASE in one way or another trace their roots back to the 1970s and the Jet Propulsion Laboratory. Here is the tale as best I can remember.

PCs were originally very expensive. In the late 1970s you could buy a "well equipped" Chevrolet Caprice Classic 4-door sedan for just over $4,000. In the early 1980s you could buy a dual floppy monochrome PC for about the same amount. When clone vendors entered the market you started seeing dual floppy clone PCs for under $2,000. The higher-end PCs started adding full height 10MEG hard drives to justify keeping their price so high. Eventually, you could get a clone PC with a whopping 20MEG hard drive for nearly $2000.

Once that $2000 price point for a PC with a hard drive was achieved, the PC started getting pushed into the world of business. The first thing the businesses wanted to do with it was keep things in sorted order. They heard from kids enrolled in computer programming courses that midrange and mainframe computers used a language called COBOL which supported indexed files that could be used to store invoices, payments, etc., all in sorted order, so information was quickly (for the day) retrievable. Well, the PC didn' thave that, and business users needed it. There was a non-commercial product called Vulcan written by Wayne Ratliff which kind of answered some of those needs. Ashton-Tate eventually released a commercial product named dBASE II. (They used II instead of I to make the product seem more stable. I' mnot making that up.)

Ashton-Tate had a lot of sales, a lot of money, a lot of attitude, and a lot of lawyers. This led to them believing they had the rights to all things dBASE. When the cash cow started giving lots of green milk the clone vendors piled into the fray. Ashton-Tate let loose with a blizzard of lawsuits trying to show it was the meanest dog in the junkyard. The clone vendors quickly got around the dBASE trademark infringement by calling their file formats xBASE. (Some called theirs X-Base, others XBase, etc.)

Times and public sentiment turned against Ashton-Tate. The people who spent many hundreds of dollars for these tools and even more money for some of the run-time licenses which had to be in place on the machines before applications written with the tool wanted a standard. When they were finally fed up with Ashton-Tate or one of the clones, they naively believed it would be like those old COBOL programs, recompile and run. Silly customers. This was the peak of proprietary software (the height of which turned out to be Microsoft Windows, which even today is considered one of the most proprietary operating systems running on a PC architecture), and there was no incentive for any of those receiving run-time license fees to agree to a standard. Well, no incentive until the business community as a whole deemed the fees they charged too high.

When the price of a run-time license reached hundreds, the business community cried foul. When the memory footprint of the run-time meant you couldn' tload network drivers or other applications in that precious 640K window accessible by DOS, dirty laundry got aired rather publicly.

Vulture Capitalists, always sniffing the wind for dirty laundry and viewing it as opportunity, started hurling money at software developers who said they could write a C programming library which would let other programmers access these files without requiring a run-time image or license. The initial price tag for those libraries tended to be quite high. Since there were no royalty payments, the developers and the Vulture Capitalists thought the "best" price they could offer was something totalling about half of what the big corporations were currently paying for development + run-time license fees. For a brief period of time, they were correct. Then the number of these libraries increased and the price got down to under $500 each. The companies vending products which required run-time license fees saw their revenue streams evaporate.

The evaporation was a good thing for the industry. It allowed Borland to purchase Ashton-Tate in 1991. Part of the purchase agreement appears to have been that Ashton-Tate drop all of its lawsuits. After that committee ANSI/X3J19 was formed and began working on xBASE standards. In 1994 Borland ended up selling the dBASE name and product line to dBASE Inc.

The standards committee accomplished little, despite all the major vendors participating. More of the data file formats, values, and structures were exposed by each vendor, but each of the vendors in the meetings wanted every other vendor to adopt *its* programming language and methods of doing things so it would be the first to market with the industry standard.

There are still commercial xBASE vendors out there. Microsoft owns what was Foxbase. dBASE is still selling products and migrating into Web application creation. Most of the really big-name products from the late 80s are still around; they just have different owners. Sadly, Lotus Approach was dropped by IBM and not resurrected when it came out with the Symphony Office Suite.

I will hazard a guess that some of the C/C++ xBASE programming libraries from my DOS days are still around and being sold by someone. That would make sense now that FreeDOS is starting to get a following. Not quite as much sense given all of the OpenSource C/C++ xBASE libraries out there, but the old commercial tools have a lot more time in the field,and should therefore be more stable. I know that Greenleaf is back in business and you can probably get a copy of GDB (Greenleaf Database) from them; I just don't know what platforms they still support.

There is a lot of history and folklore surrounding the history of xBASE. You could probably make a movie out of it like they made a movie out of the rise of Microsoft and Apple called "Pirates of Silicon Valley" in 1999. You can piece together part of the history, at least from a compatibility standpoint, by obtaining a copy of *The dBASE Language Handbook* written by David M. Kalman and published by Microtrend Books in 1989. Another work which might be worthy of your time is *Xbase Cross Reference Handbook* written by Sheldon M. Dunn and