Cocoa Programming for Mac OS X
Third Edition

Aaron Hillegass
ISBN-13:978-0321-50361-9

Read: 2008 November 1 to 2009 June 30
Reviewed: 2009 September 25

The basic plan at Barely Works Technology is to build my own radios and implement signal processing algorithms for them in software that I write myself. This is complicated by the fact that I'm doing it in a Mac or linux environment, meaning that the existing ADI tools, which are strongly PC-only, won't work. Further, the DSP-10 is the one RS-232 device whose delivered software (as of 3.80) won't work under Virtual PC through a Keyspan adapter.

So....

I need a few new programs. And for my self-educational purposes, I'd like them to be original. I need a host interface from the Mac (in some form) to the DSP-10. I need a way to build for the DSP-10, and I need a way to load that code. First I tried without training to write up a simple program that would allow me to hook into some C++ code that I already had. When this failed in miserable frustration, I started looking around for documentation. When that was incomprehensible without some basic knowledge, I started looking for a tutorial. Paul Williamson recommended Hillegass. There was a third edition on the way that would cover the migration from Xcode 2 to Xcode 3, but the second edition was all that was available at the time. I got it and started into it doing the examples in Xcode 2. This beginning effort was stalled out in the spring / summer of 2008 by my son's graduation from high school. By the time the festivities were over, third edition was out and my pre-publication order had arrived. It had been long enough that I just upgraded Xcodes and started over.

Since I was going to have to learn Xcode anyway, I decided to go ahead with "Objective-C" and "Interface Builder" so that the results would be Mac programs with the possibility of a "comfortable" user interface and graphics output, not just line commands with tables of numbers, an older paradigm with which I am quite comfortable as a developer but not so much as a modern user, and which might be incomprehensible to other potential modern users.

Over twenty years ago I learned the basics of astrodynamics from Bate, Mueller, and White "Fundamentals of Astrodynamics" (1971). I special ordered the book at a nearby bookstore (long before amazon.com or even the internet) and when it came, I started into it, working all the exercises. That was the planned approach here too. I paused all BWT activities to crank through Hillegass, hoping to achieve the competence to continue on those needed programs mentioned above.

The read dates above indicate that this process took eight calendar months. I took perhaps twenty pages of notes, not counting the ones I made in the book itself (and not counting a lot of e-mail and browsing for answers). I did indeed do all the exercises, particularly the challenges. They were painful, but it really is "no pain no gain" in this sort of business.

Hillegass, who is associated with the "Big Nerd Ranch" where I could go to camps and attend live tutorial sessions along these lines, does a remarkable job of leading the beginner through things he/she can do, giving them what they need (just barely, usually) without failing to challenge them to move forward as they must. He begins with history and conventions and ends the first chapter with a must-read section called "How to Learn." He recommends things like ten hours sleep a night, claiming that "Caffeine is not a substitute for sleep."

One story is worth quoting exactly:

"I used to have a boss named Rock. Rock had earned a degree in astrophysics from Cal Tech and had never had a job in which he used his knowledge of the heavens. Once I asked him whether he regretted getting the degree. 'Actually, my degree in astrophysics has proved to be very valuable,' he said. 'some things in this world are just hard. When I am struggling with something, I sometimes think "Damn, this is hard for me. I wonder if I am stupid," and then I remember that I have a degree in astrophysics from Cal Tech; I must not be stupid.'"

I have similar credentials that were similarly helpful in this process.

My history with computers goes back to a three-bit boolean machine that my parents got me as a gift in the 60s. It was an education in Boolean logic and could be programmed with long and short soda straws to do several operations, the most complicated of which was the game of Nim in which two players add one or two attempting to get to (and prevent the opponent from getting to) exactly seven.

I was the first kid in my high school to have a calculator, a Texas Instruments SR-10 (slide rule). (See http://www.vintagecalculators.com/html/texas_instruments_sr-10.html .) I programmed the SR-10's big brother, the TI-59, to do astrodynamics (see above), then, based on that work, the Timex Sinclair 2000 (with 16K memory expansion, was ZX-81 kit). That was my first use of Basic. During my engineering degree I learned Fortran, and with my first IBM PC got into Basic, Forth, and even 8086 assembly. Later, like most of today's old timers, I branched into C and even later C++. I have been introduced to Java, Python, and Pearl but have not done much with them.

From that background, the jump to Cocoa and IB was big, possibly the biggest one ever. It has been very hard coded into me, for example, that an unresolved symbol reference won't even compile, much less link, much less run. If Objective-C, trying to execute some GUI tries to call a method or a class that *doesn't exist* it will just happily go along without even burping and do nothing. I am now beginning to understand why this is a good idea and why the vast sea of

classes and methods that are provided by the system (with programmer level documentation only) are a good idea, but it took a three day weekend, and a lot of head bashing on a challenge exercise, and a lengthy e-mail exchange with Paul to grasp that for the first time.

The compiler does give a warning that is easy to miss that your object (that might not exist simply because you misspelled it) might not exist. You only get warnings when you build from scratch so I have learned to do a "clean" and rebuild looking for just such typos.

This really is a new world, but now I feel competent to write at least a simple program with a very Mac-like user interface. An unexpected byproduct of all this education is that I now understand why so many things in Mac programs (like searches, pull down menus, main menu contents, and so forth) seem to have the same quirks as each other. It's because they're all provided by this recommended environment. They are all the same code. Indeed, Interface Builder has recommendation markers to help you center things or place them the proper distance from edges when constructing a GUI.

Not everything I wanted to learn was in here. For example, I'll be writing a DSP chip compiler and the output from that will be hex code, ala the olden days. I'm going to have to mess around to get that to work right, that is, just put out a flat file and nothing else, and not have the object archiving system or all the standardized, "under the hood" data handling stuff mess that up.

There were beginner exercises in graphics, but nothing so "low level" as to show me how to just generate up my waterfall display from data pixel by pixel.

And of course nothing these days understands about RS-232 (not even Office Depot) or even serial ports in general so there are some known unknowns getting from here to where I want to be. But now I'm ready to start and know where to start looking for help. (The Apple developer's documentation on the web and Google, of course. Both were moderately helpful here.)

So, being new to GUIs, I didn't really feel qualified to know what I would and wouldn't need, so I marched through everything with equal rigor and vigor. At this point I nearly get most of the basics of Objective-C and even have a copy of the 2.0 Language guide. Warning to the guy who came up the way I did, up through C++ then jumps into Objective-C: There are things about the syntax that just don't make sense at first. Objective-C is not a cousin to C, it is really another language. That being said, there are easy ways to use C and C++ with it, so at least the tools know what's going on.

Hillegass takes you through examples that illuminate the new paradigms, including the standard "data / view / controller" model. I still don't think I understand this enough to do original work, meaning that my next applications will be based on exercises in the book that do. Every place where a concept needs illumination, like events or controllers, drag-and-drop, or custom views, garbage collection, core animation, and interfaces to open source applications, how to hook up to IB constructs, or do them in code (I'm just scanning through reproducing about a third of the

table of contents here) Hillegass leads you through a discussion, then an exercise, then possibly a challenge. My standard response to this (hey, I work at JPL, I'm not stupid...) is to read through the chapter carefully saying, "Yeah, yeah, this is all obvious" then start into the exercise and realize that I haven't the slightest idea what I'm doing or how to start, then page back through this chapter and portions of several past ones, sometimes opening up past exercises simultaneously to look for hints. Often I'd end up going out to the web (as mentioned) or to Paul my consultant before finally getting the thing going. Sometimes I'd even zip up my project and have Paul mess with the totality of it. One time it even took *him* several hours to figure out what was going on, saving me several dozen hours. I was ready for the ahah moment nonetheless.

The worst was a case of compound new features. I had something that I was sure I had instantiated that I was sure was spelled right, but the way the system works it had not, in fact, been instantiated or initialized *at the time I thought it was being called* so my program just --- did nothing --- without signal, warning, or (what I would have expected) crash and core dump. This was totally befuddling. I still don't know, without looking it up, exactly how to get things instantiated and initialized before I try to start using them, or what the order of things performed by the invisible event loop really is.

And that reminds me of the other big paradigm shift here. I'm used to programs that you write, that have beginnings (load, initialize, run, clean up, unload) and those programs, if they need input or have output, will query for or produce it. That's not what happens here. A Cocoa application is a big event loop that's provided to you. *It* calls *you* when it wants to (as a result of a user event like a key or mouse click, for example), assuming you've spelled your method names right. Writing like that takes some getting used to too. No wonder allocation and initialization, deallocation and garbage collection are so important.

And the Interface Builder? Does it compile from my drag and drop instructions? Is there code I could look at to see what I've done and what has been made of it? Well, kind of and no, respectively. So there's a big piece here that I can't really --- disassemble --- to look at. I just have to do it right in the first place. At least this, the user interface, the art of the presentation, is not the area where I want to major. I only want to be competent enough there to get some useable human interface to my code.

And who knows, I may in fact want to set up a database with pictures and behaviors in it. I don't think so now, but it's available and maybe it will seem like just the thing at some point. So, I need to know what I'm doing.

Unlike the astrodynamics book in which I didn't get far into Chapter 4, I read every word of Hillegass 3 and did every problem. This took 105 hours and 15 minutes of my BWT time, more than I wanted it to by a factor of three (or thirty!). That's for 418 pages of material (not counting the index). I'm already wondering what else I might have to learn to carry on.

As always, you start out with an idea of what the computer should do and you end up just accepting whatever it will let you do. Sometimes that's a subset of what you wanted, sometimes its completely different, but it is never the original dream.

I miss the days when you could understand microprocessor behavior down to the gate level and actually take out a volt meter and just measure things. We're a long, long way from that now.

A future issue here is that these tools advance faster than I'm going to be able to use them. That was one of the reasons I resisted doing this so long in the first place. The present exercise notwithstanding, I am quite resistant to spending all of my precious BWT time keeping myself up to speed on paradigms and tools. I might just have to freeze where I am today, with 2005 hardware and 2008 software. Or maybe it will be stable long enough that this won't be a problem. Deprecation is the enemy. "Engineers, they're always changing everything!" (Bones McCoy, some Star Trek movie)

Hillegass understands at the end that the graduate from this book now has the tools to go continue to learn what he / she needs to learn. He also realizes that they don't know it all (they don't even know most) at that point but that they are ready to tackle what is before them. (This, too, is very unlike learning assembly, where the knowledge was small and finite and you could master it all.)

He recommends starting to write your own applications (preferably applications that someone else will use) as soon as possible. Unfortunately, work and other discretionary activity interfered and I am only now beginning to see my way back to that first application that I was doing a year and a half ago that led me into this learning experience. Still, it has been worthwhile. Just like the day that I started a download that was going to take until after lunch at 2400 baud and instead went out and bought a new modem (33.6 K at the time), installed it, and finished the download before lunch, it was necessary for me here to make this quantum jump ahead into the world of development for today's computers, the computers that I actually use today, and to get beyond development boards and linear programming.