Chapter 14
Wombats and Eggs

A few weeks later, the Menu Builder was finished, but then I had a new problem.

I entered Heindel's office, and after the usual "Hey Heindel-man" greeting, I got to the heart of the matter. "Dave wants me to add multiple windows to READ," I said. "I was hoping you could give me some advice." Heindel had his desk positioned so he faced the door. I walked left to stand near the end, next to his black guest chair.

Heindel's face tracked me as I moved. At the word "READ" his cordial expression suddenly turned sober. "Be very careful," he said. "And don't break anything."

I shook my head. "Thanks," I said flatly. At times, Heindel had the wit of a first-grader.

Heindel turned back to study his computer screen. "Don't mention it," he said. He then squinted at the screen as if the conversation was over.

There was a stuffed Cleveland Browns football placed prominently on Heindel's desk. I contemplated whether whacking another employee aside the head with it would be against company policy. *Probably not.*

Heindel turned back to look at me and chuckled. "I'll give you a walk through of the code for READ, but I warn you, it ain't pretty. And be prepared—every time I touch that code, I break something."

"Really?" I asked, easing myself into the guest chair. *Why can't it ever be easy?* The chair caught on the carpet. I fought to bring it closer to Heindel's desk.

"Really." Heindel cupped a hand and jerked it forward in the air as he talked; one jerk for each point. "Try as you might, no matter what, if you change anything, somewhere, somehow, someone's dBASE program won't work right anymore."

Not what I wanted to hear. "Well...I don't know if I want to…."

He tried to reassure me. "You're the creator of the Screen Painter, you can do it. Just be prepared to test. **A lot**."

Great. What have I been thrown into now…?

* * *

Though only a single command, READ was one of the most intricate and convoluted commands in our product. The purpose of READ was to animate an interface. A user, when creating an application's interface, typically defined different elements of it using a list of @SAY/GET commands. At the end of the list, they would issue a single READ command to make the whole thing live. A simple interface program written in the dBASE language might look like this:

```
USE dvds
@3, 13 SAY 'Title:' GET dvds.title COLOR gr+b, r/w
@4,13 SAY 'Description:' GET dvds.description COLOR gr+b,
r/w
READ
```

This program makes use of a database (dvds) created to hold an inventory of the DVDs I own. When run in FoxPro, it would present a blank screen with the string "Title:" on one line and next to it a rectangle of text that displays the actual title of one of the DVDs in my collection. The string "Title:" is the SAY portion of the command in action and the title presented is the GET portion. The user of the program could change the value presented in the GET portion. Directly beneath the "Title:" line would be a similar line for "Description:" and a corresponding changeable (GET) rectangle of text next to it.

That is an example of the interaction READ allowed. It also allowed for more complex interface elements, like push buttons, radio buttons, and checkboxes—the sort of things anyone who uses a computer interacts with regularly.

My initial naive assumption was the code to implement the READ command would be relatively straightforward. All it had to do was handle the interaction of a few interface elements. We had code in our Control and Dialog Managers doing similar things and it was all accessible to me. Easy to read, easy to understand…it didn't take much studying of the code to comprehend—at least in part—what was going on. I could even make a change if necessary. I couldn't imagine READ being that different. How bad could it be, really?

Heindel's initial walkthrough hinted at an answer and a short while later—when I was sitting in my office with the code for READ printed out—I knew for sure.

The answer was "really bad."

In my courses at college my instructors drilled into my head the proper steps to follow in writing—what they called—"structured code." At the time, some of the rules they gave me seemed a little overzealous. A few of the important ones went like this:

1. A single subroutine (a small self-contained portion of code) should essentially have one purpose. This is the programmer's version of "A place for everything and everything in its place." Create a routine to do one thing, and use it for just that one thing. If you need it to do something slightly different, write another routine.
2. Every routine should have only one entry point and one exit point. This means as the computer makes its step-by-step way through the code it should have only one place to begin a routine and only one way to finish it. Just like reading a book…you start at the beginning and end at the end. No shortcuts. (Especially with this book!)
3. If you find yourself writing the same lines of code in a subroutine more than once, you should create another routine that consists of those lines of code and call it from the original routine. This is essentially like speed dial on your phone. If you dial a number regularly, you should probably put it on speed dial.
4. Document your code. This means put comments around any code you wrote to tell the next person what you were doing. Anyone who has put together a child's toy knows this one. The better the instructions, the easier it is to understand. For programmers, comments are the roadmap of another person's mind.

Unfortunately, the only thing READ had in common with this list of rules was it didn't follow any of them.

At an earlier time, Heindel gave me a way to maximize the amount of C code I could get printed on a single sheet of paper—using a very small font, it put out around seventy five lines of code per sheet. Even printed that way, READ spanned nearly twenty pages of text. And the code itself read like a horror novel.

It contained one huge decision structure (called a 'case' statement in C lingo) with multiple entry and exit points and several looping structures (constructs that tell the computer to do a section of code repeatedly) that spanned all twenty pages. Nearly everyone who was hired before me touched it (indicated by their initials near individual lines of code—with no additional comments), but none of them would ever claim responsibility.

The READ code was what my friend Rusty would call a "can of wombats." Now I was going to be right in the middle of it.

* * *

In the first version of FoxPro, a user could only define their interface (@SAY/GET) elements in a single window. This behavior emulated what FoxBASE+ and dBASE IV did, so there was no immediate pressure for us to change it.

Neither of those products were really windowing environments, though. dBASE IV allowed for the definition of windows, but the primary interface of that product was still very much like its predecessors. Each tool—their editor, their BROWSE command, their Report Writer—were confined to a single screen and didn't interact much with the rest of their product.

FoxPro wasn't like that. Our windows were sizable, they moved, and they overlapped each other. Our tools lived inside windows of their own. Having a READ command that didn't allow our users to provide **their** users with a similar experience just didn't seem right. My mission was to give them that ability.

Of course, feeling confident of my coding skills, I thought I could clean up the READ code a little before I added the changes to make it support multiple windows. In my first glimpse of the main READ routine, I saw dozens of places where the exact same lines of code were written. If I could break those places out into separate routines, I'd not only get my feet wet in the ocean of code that made up READ, but I'd help the code itself. Shrink it. Make it a little more structured.If that worked out, I could press on to make it obey some of the other rules of structural programming. When finished, READ would be a concise, well documented—purely structural—routine. From there it would be easy to plug in my multi-window changes.

So, that's what I did. I created a few new routines to replace the duplicate code and made the appropriate changes in the READ code to call them. I built a new version of the product and did some testing.

READ worked fine. I was confident it would, though. The changes I made were about as innocuous a change as could be made. Usually, if your product will build with them in, they work. After a little more testing, I checked in the first round of my new and improved READ.

* * *

The next day Janet was at my door. "I'm seeing something weird," she said. "Come take a look at this."

I followed her to her office. On her computer's screen was a relatively simple READ in action. It had a couple of database

fields displaying their data in GET rectangles, some text, and a couple push buttons.

"Now watch this…" she said. She proceeded to hit the tab key a number of times. As she did this, the color of one GET at a time would change to show it was the currently selected one. When she reached a particular GET, instead of the entire rectangle changing color, a cursor appeared within.

"Yeah?" I said.

Janet looked at me. "Well, yesterday that GET would be completely selected. Today it is just showing a cursor. Heindel said you were working on READ now so…"

I hooked my fingers on my belt loops and looked out Janet's window. She was located in the back of the building. Beyond a small stretch of asphalt, she could actually see trees and houses. I looked back at the screen again. "And it didn't do this before?"

Janet shook her head. "Do you want the code to reproduce it with?"

Not really. "OK…."

I took a disk with me back to my office. After putting Janet's files on my machine, I brought up my version of FoxPro and ran her program. I saw the same behavior I saw in Janet's office.

I next took the steps necessary to "back out" temporarily the changes I made the day before, built a new product, and tried Janet's program again. Just like she said, it worked differently.

"Well for all the—" I shook my head. "How?"

I spent the better part of an hour figuring out "How."

Every GET element in a READ may have a number of "clauses" associated with it. In our manuals the description of the syntax for the form of GET Janet was using (and there were many others) went like this:

```
@ <row, column>
GET <memvar> | <field>
[FUNCTION <expC1>]
[PICTURE <expC2>]
[DEFAULT <expr1>]
[ENABLE | DISABLE]
[MESSAGE <expC5>]
[RANGE [<expr2>] [, <expr3>]]
[VALID <expL1> | <expn4> [ERROR <expC6>]]
[WHEN <expL2>]
```

Every line after the GET <memvar> line was what we considered a "clause" and every one of them in some way could affect the behavior of a particular GET, if they were included. With my changes I

inadvertently altered the behavior of one of those clauses. Once the problem was known, though, the fix was relatively straightforward. One of my new subroutines needed to act slightly different when it was called from one place in READ than it did in all the other places it was called. It wasn't a big enough difference to elicit an entirely new routine, so I pulled another tool from the coder's toolbox. I made it so I could send a flag (a simple 'YES' or 'NO' value) into the routine. A 'YES' meant it was called from the place that needed to act differently, a 'NO' from all the others. I made that change to the READ code, tested it, and checked it in.

Later that same day, the writer named "Dave" (Venske) arrived at my office door. In his late thirties, he had full, slicked-back head of hair, and an easy smile.

"Janet says you're in charge of READ now," he said. "I have something that's acting weird for me today."

"Do you have the code?"

He handed me a disk and gave me a short explanation of the problem.

"Let's give it a shot," I said as I copied his dBASE code to my machine. "Maybe I've fixed it already." *Hopefully the changes I made this morning for Janet*…. I ran his code and GETs filled my screen.

He chuckled. "Nope. Still there." He pointed a finger, indicating the problem.

I looked heavenward, frowning. "OK…I'll take a look at it."

An hour later, I figured out his problem. A different clause and a different side effect of the same trivial changes. I made another—more complicated—fix.

The next day, another person appeared at my office door. "I heard you're in charge of READ…."

"Yeah…?"

After about a week of similar occurrences, I finally had my initial changes to READ working. I also came to realize the code for READ was not just a "can of wombats," it was what the senior members of the team called "a tower of eggs." You can construct the tower once, but after it is up, it's best looked at, and never touched again.

I also decided not to press ahead with any more READ renovations. I'd just make the multi-window additions Dave wanted and get out. The odds that I could make those changes without breaking something were already incredibly slim—why push my luck?

* * *

I managed to finish the multi-window READ changes by the end of September, but by then, Dave had another list of enhancements he wanted added to READ. Those changes came with another level of complication.

"Look at this!" Dave said after he hauled me into his office. He brought up the Screen Painter and started pointing at his screen. "I can't seem to find the places in the Screen Painter where I can set a control's DISABLE attribute. Where are they?"

The answer to that question was easy. "They're not there yet," I said.

The corners of Dave's mouth turned down. "Why not?!"

My heart started to pound. For the last few weeks I not only had to add features to READ, I had to add a corresponding means for accessing those features to the Screen Painter as well. This round-robin approach was in direct opposition to the way I liked to work. I liked to code one part of a feature at a time and test it until I was confident it worked. Consequently, I liked to be responsible for one under-development portion of the product at a time and work on that until it was finished. Working on two large and complicated portions of the product, both undergoing frequent changes, with some interdependencies between them, but no code really being shared, was a lot to keep my arms around.

"Well…" I said. "I haven't had time, Dave. I've been working on adding it to READ first." I glanced at a set of juggling balls Dave kept within arms reach of his keyboard. Many in the development staff took up juggling as a hobby—Dave especially. I didn't know how to juggle and frankly, I didn't care to. One ball in the air alone was enough for me.

Dave studied me quietly for a few moments. Then, out of the corner of my eye, I saw Heindel pass by Dave's door.

"Oh, David!" Dave said to Heindel.

Heindel, who had just made the turn toward developers' row, quickly turned around and walked back to Dave's door. "Yeah, Dave?"

"How's your stuff coming?" Dave asked.

"Pretty well," Heindel said, shrugging. "Should be done soon."

"Good…good…" Dave said, drumming his fingers on his desk. "There are some READ changes that need made."

Heindel glanced at me. "Oh?" He thought he'd left READ behind.

"It **is** your bailiwick. Kerry has Screen Painter changes to make."

I remained silent and expressionless, but inside my emotions were mixed. Part of me felt like I failed for not being able to handle both areas concurrently, but the other part of me—the larger part—was feeling really good.

"OK," Heindel said hesitantly. "I'll work on READ…."

Dave waved us away and I followed Heindel back to his office.

"Dave wants me working on READ," Heindel said, mostly to himself.

"Sorry," I said. And I was. I hated others having to pick up my mess.

Heindel acquiesced, flipped up a hand. "Dave wants me working on READ. Fine…fine…."

Heindel was a proficient juggler. For that, I was grateful.

* * *

The following weekend I took our family friend, Marlene, in to see where I worked for the very first time. Few people were around that day, but in the middle of the tour, Dave strolled out of his office holding a bag of popcorn.

"Hey Dave," I said. "This is a friend of mine—Marlene."

Marlene smiled. "Yes, I sort of motivated Kerry into the computer field…into programming."

"Oh really?" Dave said as he stuck out his empty hand to clasp Marlene's. He bounced a little and cackled loudly. "Maybe I should hire you—to motivate him now!"