



## *Small Things | Loosely Joined*

### **Dr. Don Levan**

President, Vanguard Custom Software  
NY State Licensed Clinical Psychologist  
FileMaker Certified Developer

### *The Problem: Programming Is Hard*

Our goal is to efficiently and reliably create in code that which we, or our clients, can imagine.

We are modeling complex worlds and process in memory (our memory). Unfortunately, human beings are ill-equipped to deal with such complexity. Our short term memory is limited, fragile, and constantly replaced. We can only attend to one thing at time.

Applications quickly grow so large that they overwhelm our ability to maintain coherent mental models about how they work. In addition, scripts often become non-orthogonal (or interconnected) and difficult to change.

Our challenge is to create software that works, is easy to extend and modify, and is understandable when we, or a peer, returns to it in the future.

Meeting this challenge requires that we solve several significant problems:

- 1) How do we create complex applications out of simple parts?
- 2) How do we separate or decouple the parts so that changes (and bugs) do not cascade through our applications?
- 3) How can we keep our applications flexible enough that we can quickly add new features and respond to shifting requirements?

Fortunately for us, several generations of hackers and scientists have developed philosophies and practices to address these problems.

### *Goals of this Presentation*

In this session, we will review the philosophy and practices developed to improve software quality, increase code reuse and portability, and increase programmer efficiency; and identify the ways those ideas can be applied to improve our programming in FileMaker.

*“The only way to write complex software that won’t fall on its face is to hold its global complexity down.*

*Build it out of simple parts connected by well-defined interfaces, so that most problems are local and you can have some hope of upgrading a part without breaking the whole.”*

*Eric S. Raymond, The Art of Unix Programming*

## Key Points

- Make each script and function do only one thing.
- Use as few lines of code as possible.
- Name scripts to clearly describe their purpose.
- Do not repeat yourself (DRY).
- Keep your variables as local as possible (SHY).
- Pass data in and out as script parameters (Tell the other guy).
- Break complex scripts into disconnected layers.
- Expect the output of any script to be used as the input of another.
- Separate out the data from the program logic.

## {the Unix Philosophy}

Unix is the most relevant of the other programming traditions to FileMaker developers. Unix was developed 1969 and has enjoyed a myriad of uses.

Doug McIlroy (As quoted in Raymond 2004), one of the founders of Unix, summarized the Unix tradition in this way:

- (i) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
- (ii) Expect the output of every program to become the input of another, as yet unknown, program.
- (iii) Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.

In this presentation, we will examine three tenants of the Unix philosophy most applicable to FileMaker development : create Small things, that are loosely joined; if you must fail, fail loudly and as soon as possible; and fold knowledge into data so program logic can be stupid and robust.

## Create Small Things | Loosely Joined

### Small things are compact, transparent, and do one thing well.

A small program is one that you can hold in your memory. A program or script is transparent when you can quickly understand its purpose, intent, and mechanics.

A script, custom function, or calculation that does only one thing is easy to test, and easy to understand. If each of your scripts does only one thing, and is clearly named, then you can use it in a routine way and your code is self-documenting.

Improve transparency by flattening nesting and indentation, using one screen or less of code per script (< 50 lines), and naming scripts to clearly describe their purpose.

### Loosely Joined programs keep it “Dry, Shy, and Tell the Other Guy”

Don’t repeat yourself. A program is “Dry” when every piece of knowledge has a single, unambiguous, and authoritative representation within a system. This includes script logic, data, constants, and comments.

Keep your code shy by eliminating dependencies on other code. The more you can do this the more you can make your code context free and protected from other changes in the system. This means keeping your variables as local as possible. When you do use globals, create them and destroy them close to their use as possible, and give them a clear name to ensure that you will not inadvertently change their function or data in other scripts.

Utilize script parameters to pass the data in and out (Tell the other guy), and expect the output of any script to be used by another. as of yet unknown script.

The Pragmatic Programmer describes it in this way, “When components are isolated from one another, you know that you can change one without having to worry about the rest. As long as you don’t change that component’s external interfaces, you can be comfortable that you won’t cause problems that ripple through the entire system” (p. 35)

Note. The quote at the head of this section is from Dave Thomas and Andy Hunt, IEEE Software, 2004

## The Unix Pipe & the FileMaker Pipe

In the Unix tradition, scripts are connected to each other in complex “pipelines”. Every Unix script has access to three streams of data via which they pass information: Standard Input (stdin), Standard Output (stdout), and Standard Error (stderr). The Pipe (the vertical separator “|”) connects the standard output of one program to the standard input of another.

In FileMaker, we have access to the same three streams through the use of script parameters and script results. Get (ScriptParameter) allows us to pass information through the input stream. Exit Script and Get (ScriptResult) allows us to pass back the script’s output and any errors.

There are several established techniques for passing multiple parameters as name/value parameters between scripts. I am partial to the custom functions developed by Geoff Coffee and Jesse Attunes at Six Fried Rice. Daren Terry also reviewed many of the most common techniques at a session at the 2010 Portland Pause on Error. See the references on the last page of this handout for more information. You are encouraged to identify and use a method that works for you. Use it on every project.

## Reduce “Technical Debt” by refactoring

Ward Cunningham, creator of the first wiki, coined the term Technical Debt to describe the problem when you have to quickly add functionality to a system. Cunningham writes,

Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly

with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on the debt. Entire engineering organizations can be brought to a standstill under the debt load of an unconsolidated implementation.

You can reduce technical debt in your projects by refactoring to make code more compact, transparent, and loosely joined. Martin Fowler, describes refactoring as a process of “restructuring an existing body of code without changing it’s external behavior.” The key to refactoring is making one simple change at a time, and then testing to ensure that the system is still functional.

## *Fail Loudly, and as soon as possible*

If you Set Error Capture on, and leave it on you are simply suppressing the reporting of any errors. While this can be comforting, it provides a false sense of security. It is far better that your users receive (and notify you about) any error messages.

Set Error Capture (Off). Leave it off, except for the few script steps when you need it on. In those cases, trap for and handle any errors. Report the errors upward through your script stack by ending every script with an Exit Script step in to which you pass any errors that were found.

Take action on the most important errors by showing the user a custom dialog, writing the error to a log table, or emailing the developer or system administrator.

## *Fold knowledge into data, “so program logic can be stupid and robust”*

Wherever possible, make a distinction between the program logic and the data. Data changes, code is more static. Data is easier to understand, code is more abstract.

The more you can separate out the data from the code, the more portable and reusable your scripts will be. FileMaker gives us many great tools to separate knowledge data from logic. These include the Evaluate and Let functions, the Set Field By Name script step, the GetField function, globals Fields, and global variables. You can also create user preference and session tables, and store constants in custom functions.

## Programming Resources and References

### Books, Webpages, and Articles

Raymond, Eric. S. (2004). The Art of Unix Programming. Addison-Wesley.

McConel, S. (2004). Code Complete 2, A Pratical Handbook of Software Construction. Microsoft Press.

Pine. C. (2009). Learn to Program (2nd Edition). Pragmatic Bookshelf.

Thomas, D. and Hunt, A. ( 2004). OO in One Sentence (Keep it Dry, Shy, and Tell the other Guy). IEEE Software, May 2004. [http://media.pragprog.com/articles/may\\_04\\_oo1.pdf](http://media.pragprog.com/articles/may_04_oo1.pdf)

Hunt, A., and Thomas, D. (2000). The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley.

Refactoring.Com. Martin Fowler . <http://www.refactoring.com/>

Cunningham, W. (1972). The WyCash Portfolio Management System. In this article Ward Cunningham first described the concept of Technical Debt. <http://c2.com/doc/oopsla92.html>

### Parameter Passing Methods

Passing Multiple Parameters to Scripts, by Jesse Antunes, Six Fried Rice, <http://sixfriedrice.com/wp/passing-multiple-parameters-to-scripts-advanced/>

FileMaker Dictionary Functions, by Jesse Antunes, Six Fried Rice, <http://sixfriedrice.com/wp/filemaker-dictionary-functions/>

Parameter Pandemonium, A presentation by Daren Terry at the Portland 2010 Pause on Error Conference. Sample File, [http://pauseonerror.pbworks.com/f/Parameter\\_Techniques.fp7.zip](http://pauseonerror.pbworks.com/f/Parameter_Techniques.fp7.zip)

### The “Craft of FileMaker” Workshop

Interested in learning more? Join Don Levan and Ernest Koe (of the Proof Group) in a three day seminar on the craft of FileMaker Development.

Hosted in New York City in October, 2010, this workshop will feature a combination of lectures and small group experiences to help you dramatically improve your abilities to understand the problems to be solved and design and code effective solutions.

For more information send an email to: [craft\\_seminar@vanguardcs.net](mailto:craft_seminar@vanguardcs.net)



**V**anguard Custom Software was founded by a psychologist with a passion for solving problems.

Our mission is to create simple applications which solve complex problems. We incorporate software development best practices, interaction design, and cognitive psychology to create data driven applications and websites that are efficient for business and easy for people. We employ only FileMaker Certified Developers.

Please contact us for assistance on your next project.

**Don Levan, Psy.D.**

President, Vanguard Custom Software  
NY State Licensed Clinical Psychologist  
FileMaker Certified Developer

917 842-2911  
[www.vanguardcs.net](http://www.vanguardcs.net)  
[don@vanguardcs.net](mailto:don@vanguardcs.net)