Coding Samples

Contributors: *Dan Baer, Gary Baanstra, Ben Bishop, Scott Jones, Greg Kreis, Michael L. Poxon, Jim Self, Kevin Smith, and Leane Verhulst with editing and introduction by Valerie J. Harvey, Ph.D.*

Introduction

Online discourse is an important realm for exploring technical topics. As I promised earlier in the year, I want to include in *M Computing* a better interface to the interesting technical interchanges that take place on the Internet and at Web sites. To initiate this I have selected some topics from recent M-List traffic and have sought a format that will treat a collection of excerpts as a contribution to *M Computing*. This approach will provide some print recognition to those who take the time to share their technical insights and engage in such discussion. (My ulterior motive: I would like to encourage eventual technical article contributions to be published in *M Computing* from some of these folks!).

For this issue the thread topic is "coding samples." The thread focuses on editors used in programming and on routine format and style. There is a useful treatment of handling the dot syntax in nested looping. Editor's notes are enclosed in square brackets [ ]. Some minor editorial changes and corrections have been made. The opinions expressed here are not necessarily the opinions of the MTA or even of the other contributors!

Format is a challenge in delivering a thread of online technical discussion. Threads are conversations—even the initial segment here refers to a preceding conversation. They have elements of "parallel distributed processing"—parts of the conversation go on independently (thus three "sub"—threads here). It is difficult to find an appropriate "linear" arrangement for this mode of presentation. I think it is important that more M users have contact with those who are exchanging technical ideas and that the authors get credit and acknowledgement in print. The representation of the "thread" intentionally has a different appearance from the full intertwining and embedding of the internet exchanges—each contributor is acknowedged as separately as is feasible for ideas and expression. The result is at the same time far less formal (given the conversational tone and cross-references between posts) than a technical article.

(1) Editors, Colors, and Rules

Jim Self: The M syntax-driven coloring that an editor like MDesktop provides can make this and many other errors/features of your source code stand out and be much more obvious. For instance, I generally set the background color on comments, quoted strings, and dot level indicators to make them noticeably different from other text.

If you haven't yet seen this editor or experimented with background color settings, I think you will be amazed at how much more readable they can make your code. Language features jump out at you that you might otherwise have to look at for awhile to discern.

Dan Baer: I sure do miss my MDesktop editor. It really impressed me after using it. I don't use it for my shareware product, since it doesn't use InterSystems products. The editor I'm stuck with now is extremely archaic—black & white, no jumping to line labels, it doesn't even tell you what row or column you're on.

Greg Kreis: I wonder if "Slick Edit" or some other universal editor might be useful? I have been hearing some coders (in C++, Java, etc.) talk about their favorite editors. Some of them I believe have "modules" that you can plug in to make the editor "aware" of the rules for what it is editing. So if we were to get someone to write an "M rules module," we might find some common ground. Anyone here ever heard of these kinds of universal code editors? [These are mentioned by Scott Jones. See comments below.]

Scott Jones: Actually, I'm using one—Epsilon from Lugaru Software. Check out http://lugaru.com to get an evaluation copy. The latest version, currently in beta test, allows you to edit files stored on ftp or http servers directly. Pretty nifty. The colorizing code is **VERY** simple to modify and is very fast (The editor keeps track of "safe" places that your colorizer has said that it can always start colorizing). The default interface is based on Emacs—but it is totally customizable. I've written an

"M" colorizing module. I've been thinking about making Epsilon connect directly up to a Visual M server so that it could directly edit routines stored in M datasets (Right now it automatically detects that "OpenM" mode should be used for *.inc, *.mac, *.rtn, *.ro, *.urp, and *.rsa files) but I haven't had any time to do so! I'd also like to make it do full syntax checking in the editor. (Right now I've got it so that it knows which are ANSI commands, functions, and special variables, as well as ISM/DSM/DTM Z commands, functions, and special variables.)

Kevin Smith: There is an excellent product called CodeWright by Premia! This editor is awesome and has the capabilities you mentioned. As a matter of fact, I have been developing an M plug-in for this product that would cover all the ANSI parts of M. By the way, you can get a trial copy of CodeWright from http://www.premia.com Of course, one downside to this method is that you must export your routine, edit it and then import it back into M, but that is how it works on the VAX under DSM so I am used to that.

(2) Loops, Dots, and Indenting

Michael L. Poxon:

```
[ snip]
  F   S SORT=$Q(@SORT) Q:SORT=""
      Q:$QS(SORT,1)'="SORT"   DO
[ snip]
```

(and to be honest, I don't know if the `Q:SORT=""` is redundant or not . . . .) It has not been redundant in any implementation I've seen and is certainly a minor irritation. I think having two `QUIT`s on the same line is **ugly**! I wish short circuit logic applied to `OR`s the way it does to `AND`s—that is, if I've got `IF expr1!expr2` and `expr1` is true, `expr2` is not evaluated—that way, we could at least write:

```
Q:SORT="" !($QS(SORT,1)'="SORT").... 
```

I guess the difference is that if an `AND` short circuits, the whole rest of the line is ignored so syntax doesn't matter.

Ben Bishop: `AND`s don't short circuit. Multi-argument `IF`s will short circuit, but that is the nature of the language (i.e., it explictly states that where "`COMMAND ARG1,ARG2`" is allowed, it is equivalent to "`COMMAND ARG1 COMMAND ARG2`") so "`IF A,B`" is the same as "`IF A IF B`" which is not quite short circuiting. `IF A&B` will evaluate `B` if `A` is false. A rather simple fix to the original problem might be to define that `$QS` (name, index) will return null for any "index" where "name" is null.

Greg Kreis: Here is another way to write this code. Mostly it is a style change. I find I can read code easier if the looping is separated from the code that you perform "once you have arrived." Also, with several dots to be put on each line, it is easy to miss by one. If you have a line with two that should have three, then the lines with three dots after it are orphaned, never to be performed. No error is reported with this kind of subtle bug.

Ben Bishop: I find that I rarely (if ever) miscount dots for indentation on a multi-level `FOR` loop; the coding makes the need for the new dot level plainly obvious (I try to start all such `FOR` loops with the `FOR` being the first command on the line).

Leane Verhulst: I also rarely miscount the dots. I find that if I use a "dot space" (.) combo, it really makes the dot structure stand out and makes it easier to see where it starts and ends. I also try to keep the line of code to 80 characters. Yes, I know that it is "wasted space," but I find it easier to read and therefore easier to maintain. (What! Maintenance?! What maintenance?)

As for separating out the code that is performed "once you have arrived," it depends a lot on what I need to do and why. If it is just a quick calculation not used by anything else, then I will keep it within the dot structure (and then I don't have to hunt for the tag). If it is something that could be "callable" then I will put it as a separate tag. In other words, it is a very subjective thing.

Ben Bishop: Without substantially rewriting Greg's "`PRINT`" function [not included here], I submit my attempt:

```
TAG      ;a tag in my program
   NEW SORT,ID
   S SORT=$NAME(^TEMP("SORT"))
   F  S SORT=$Q(@SORT) Q:SORT=""
     Q:$QS(SORT,1)'="SORT"   DO
 . S ID=$QS(SORT,4) ;could also use
     $QS(SORT,$QL(SORT)) for the last
 . I ID]"" D PRINT(ID) ;print one
     patient's information
 Q  ;all done

PRINT(ID) ;print information for one
     patient/ID
<snip>
```

Leane Verhulst: `$NAME`, `$QS`, and `$QL` are not in the version of M we are running. And I doubt if I would use them even if they were. In my opinion, the code is harder to read (and therefore harder to maintain). I also try

to stay away from indirection. It gives me a headache!!

A note about NEW. I also try to use the NEW command at the beginning of each tag. I just forgot to include it in this example. I also put the variables that I am going to NEW in alphabetical order. That way it is easier to find the variable in the list, especially if there are a lot of variables. For example, I have a large list of variables, and part of my list is: BCODE, CNT, CNTALL, CNTNEW, DONE. I now need to add the variable CNTBAD. I can see at a glance that CNTBAD is not being used, and I quickly add it to the list: CODE, CNT, CNTALL, CNTBAD, CNTNEW, DONE. I also avoid multiple NEWs within a tag.

Gary Baanstra: Actually, one thing I've always pondered on and never really came out with anything concrete is: at what point should you (if ever) stop indenting your code and create a call to a subroutine? This pretty much applies to any language. I've found at around 4 or 5 [indents] it gets hard to look at and usually call to a subroutine, although this would seem to run the program slower (albeit slightly).

(3) Tag Names, Case, Dots, and Spacing

Jim Self: Scott Jones wrote some very interesting comments from his own rules of M coding style which, although largely different from the others I have seen here, appear to be internally consistent.

Scott Jones: [Some of Scott's rules (1, 2 and 9) are referred to by Jim Self]

(1) All commands are spelled out in title case (i.e., first character upper, the rest lower).

Jim Self: [Regarding Scott Jones' Rule 1] Since M does not distinguish the case or long/short form of commands, intrinsic functions, pattern characters, etc., do what you wish, anyone can easily change it to their own preference, if only on output, as needed. I personally find MUMPS (including your example) easier to read with these in the lowercase short form.

Scott Jones: Ah—but since I always keep variable names all lower case, the mixed case for commands makes them easier to read—more like a sentence.

```
Set thearray(1)=thevalue
```

(2) All external tags are in mixed case starting with a non-% character, and all internal tags are in mixed case starting with %.

Jim Self: [Regarding Scott Jones' Rule 2] My own preference for the use of mixed case in line tags is to capitalize the first letter of a tag intended for external use and lowercase it for internal use. I don't use % in tag names. If a tag is the concatenation of multiple words or abbreviations, always capitalize the first letter of each but the first.

Scott Jones: I simply prefer having the name consistently mixed case and just add the % if I deem that the function is private (internal) for now. If I need to change it to public, it seems easier to me to find it and just remove the %'s. Your [Jim's] method also works, and the choice is mainly a matter of personal (or shop) preference.

(9) Use a space between .'s [dots]. It is easier to see the number of levels. Given the amount of traffic recently about people consistently having a problem when introduced to M because M is different from most other languages (Note I **didn't** say that it was better or worse!), it seems prudent to plan for new support programmers not knowing M at first having to maintain whatever I write.

Jim Self: [Regarding Scott Jones' Rule 9] Remove all spaces next to .'s to guarantee that the nesting level is clearly reflected in the position of the characters. I have encountered subtle errors that arose from a little extra white space between dots that were several levels deep and confused the nesting level of the affected lines. Again, this is one of those details which make no difference to the operation of your program as long as there is consistency.

Scott Jones: Yes, the important thing is consistent spacing no matter **WHAT** the language.

Jim Self: I strongly encourage the use of a source code filter to guarantee that all routines in a given shop conform to a single standard, whatever it is.

Scott Jones: Given the amount of traffic recently about people consistently having a problem when introduced to M because M is different from most other languages (Note I **didn't** say that it was better or worse!), it seems prudent to plan for new support programmers not knowing M at first having to maintain whatever I write.

Jim Self: My impression is that most other languages are different from **most other languages** so that one would often encounter this same level of problem or worse in learning a new language. I believe that one should expect new programmers to read the manuals, or at least the language reference materials, and that the simplicity

of the left-to-right rule for operator precedence would cause it to be one of the first features to really stick in one's mind. However, recent examples show that doesn't always happen.

Scott Jones: Well, most all of the ones that I have used, except for RPN (HP) calculators, Lisp/Scheme, APL and M are pretty consistent. I really did mean "most" other languages (at least in this one area of expression evaluation). This is one area where they are pretty much the same.

Think about it—C, C++, Java, FORTRAN, COBOL, PL/1, any non-HP calculator, Rexx, Perl, BASIC, Pascal, Algol, Ada, . . . all are basically the same in the order of expression evaluation. The list of languages that are NOT the same is rather few, and none of them are the same as any of the others.

Jim Self: On the other hand, if a new programmer knows only one or two other languages, they are probably C and BASIC, so one could argue that special consideration should be given to accommodating that particular bias.

Scott Jones: As I said above, it isn't just C and Basic—by any stretch of the imagination (think of all those non-HP calculator users)—I'd say 99% of programmers expect the order of evaluation to be a certain way and are surprised when they come up against M. At least with RPN, Lisp/Scheme, and APL. It doesn't even **look** like a valid expression so there is less room for confusion. **M**

—Valerie J. Harvey, Ph.D.

Dan Baer	*mtrc@mcenter.com*
Gary Baanstra	*gbaanstra@MDSMETRO.COM*
Ben Bishop	*aci@shore.net*
Scott Jones	*scott@intersys.com*
Greg Kreis	*gkreis@mindspring.com*
Michael L. Poxon	*Michael.L.Poxon@kp.ORG*
Jim Self	*jaself@ucdavis.edu*
Kevin Smith	*kjsmith@AIRMAIL.NET*
Leane Verhulst	*lverhulst@NMFF.NWU.EDU*

Advertiser Index

We appreciate these sponsors of the October issue and all the companies who support the M community through their commitment to excellence.

This index appears as a service to our readers. The publisher does not assume any liability for errors or omissions.