QUIT—The Ultimate Command

by Winfried Gerum

Winfried Gerum

No RISC—no fun. That is not murderous advice in the age of the AIDS epidemic, but a slogan advertising faster data processing. Today, assembler programmers are an endangered species. The job has been taken over by compilers (or interpreters!) of high-level languages. These compilers are not as clever as their human predecessors, and therefore they rarely use the more sophisticated instructions processors used to offer. So the idea has been to keep processors simple and stupid, with resulting processors being easier to debug and produce. As a lot of space on a chip is saved, chips can be made smaller or they can accommodate on-processor memory. Both options are good for speed. (And a stupid compiler never knows the difference!) The user or programmer gets marvelous speed at an affordable price.

What is good for hardware might be good for software, too. Never cry never. If airplanes were as reliable as a typical piece of software, no one would ever dare fly. If all software were as expensive as COBOL programs, there would be no basis for personal computing. Automating the software production process is common for us MUMPS folks. In former times, MUMPS was something like a reduced instruction set language. But it becomes ever more complicated. Most proposals *add* something to the language. Why not propose to *reduce* the language?

Looking at the VA FileMan version 17.7 package with 188 routines, 7,624 lines, and 424,872 characters, I found 24,622 commands, which fall thusly:

Commands	Percentages/Types
8,826	35.85 % SET
3,062	12.44 % IF
2,846	11.56 % WRITE
2,606	10.58 % GOTO
2,472	10.04 % KILL
1,812	7.36 % DO
1,683	6.84 % QUIT
720	2.92 % FOR
207	0.84 % XECUTE
204	0.83 % READ
106	0.43 % ELSE
24	0.10 % NEW
21	0.09 % LOCK
19	0.08 % USE
6	0.02 % CLOSE
3	0.01 % ZL
2	0.01 % HANG/HALT
2	0.01 % VIEW
1	0.00 % OPEN

A few commands make up the bulk of routines. Surprisingly, it is possible to avoid the most frequently used commands altogether. But there is one command that is indispensable now: the QUIT command. Previously this command seemed to be one of the least exciting ones. It was most memorable for being the source of the most frequent typographical error, i.e., just one SPace after that command. Another occasional booby trap has been an implicit QUIT at the end of a routine: when adding code at the end of a routine, one easily forgets to make the previously implicit QUIT explicit. Since the 1990 ANSI MUMPS standard, things have changed dramatically. QUIT can still be used in the argumentless form to terminate FOR loops, subroutines called DO or XECUTES. But in a form with one argument it is used to terminate an extrinsic $$function, returning the value produced by that function.

Surprisingly, now the only command you really need is QUIT. It is possible to write functions that contain only QUIT statements. The good old Newtonian square root algorithm

```
   ;;SQRT takes one argument
   ;;interpreted as a real
   ;;number. empty string and
   ;;negative values are
   ;;considered invalid input
   ;;and return an empty string.
   ;;end criterion of FOR loop
   ;;does not work on arbitrary
   ;;precision systems!

SQRT(X)Q:X<0!(X="") "" Q:'X 0
  N A,B
  S A=1+X/2
  F  S B=A,A=X/B+B/2 Q:A'<B
  Q B
```

may be restated with the help of two auxiliary functions as

```
SQRT(X)Q:X<0!(X="") "" Q:
  'X 0 Q $$SQ2(X,$$SQ1(X,X))
SQ1(X,B)Q X/B+B/2
SQ2(X,A)Q:$$SQ1(X,A)
  '<A A Q $$SQ2(X,$$SQ1(X,A))
```

Provided that your M implementation supports the necessary stack depth

needed for recursion, these two examples are functionally equivalent. There is no longer a SET command. The assignment is done via the parameter-passing mechanism. By the same reasoning, the NEW is superfluous, as parameter passing implies NEWing the formal parameters. FOR is replaced by recursion. DO is implicit in the use of an extrinsic function. There is no need to use IF since there are either postconditionals or $SELECT. Using $SELECT and avoiding postconditionals, we could rewrite SQRT as

```
SQRT(X)Q $S(X<0!(X=""):"",X:
  $$SQ2(X,$$SQ1(X,X)),1:0)
SQ1(X,A)Q X/A+A/2
SQ2X,A)Q$S($$SQ1(X,A)<A:
  $$SQ2(X,$$SQ1(X,A)),1:A)
```

Allowing for arbitrary long lines of codes, one could rewrite a lot of software as a sequence of functions all containing just one QUIT statement.

If we can do away with SET, IF, ELSE, FOR, and DO, then what about the other commands? There is almost no need to mention GOTO. Some already regard GOTO as a pariah. Since the introduction of the block structuring syntax, there is rarely a sound motivation to use GOTO. For the rest, let us have a short look at other computer languages: the now so popular C programming language does *not* have I/O statements. Likewise, Modula-2, Pascal, and ALGOL do not have I/O in the language definition. These languages make I/O available via library functions and library procedures.

Global variables are one of the pillars upon which M is founded. They make I/O easy. I cannot think of working with globals without SET and KILL. Working with local variables, there really is no need for these commands. So it is conceivable to have a RISC-MUMPS with a dramatically reduced instruction set. Imagine the benefits of such a MUMPS: ultra fast, super reliable, and dirt cheap.

It is unlikely that the M community will follow that path. But have a second look at software written in that style. The principles of "Higher Order Software" (as stated in James Martin's book *Provably Correct Programs from Provably Correct Constructs*) are child's play to follow using this style. Dead code and dead variables as most <UNDEF>s can be located by automated analysis.

You probably do not intend to rewrite time-honored subroutines such as the square-root function. When writing new software, though, it is good practice to make extensive use of (extrinsic) functions: all information necessary and sufficient to produce a certain result has to go through the parameter list. There should be just one result. This result is passed as the value of a function by the QUIT command. A function should not have any side effects. This is currently not possible to achieve in certain respects: the naked indicator cannot be restored easily (NEW $REFERENCE might help) and the I/O situation cannot be restored easily. The most notorious side effect is that on $HOROLOG. If you have any remedies, please tell me.

Breaking a problem down into subproblems may lead to mapping each problem in a natural way into a function, getting your code directly from analysis. In some cases, when the analysis is already available, you just map the analysis into code.

If you think this idea is impracticable, have a look at the following two nontrivial examples. A function to transform hexadecimal numbers into decimal format appears as figure 1. Typically, such an algorithm does not take negative numbers or fractions into account. This example also has a shortcoming, as some valid hex strings cannot be converted to decimal numbers (try $TR($J("",255)," ","A")). A really good function has to address this problem. With the 1993 M standard we will have error processing to handle these exceptions easily. A function with illegal input should produce *no* value. The functions presented here give an empty string on illegal inputs. This is only because standard error processing has not yet been established. The main function (i.e., the one to be called from outside) always has to check for illegal input. The auxiliary functionesimals (see planetesimal in Webster's New World Dictionary) may do without such checks for performance reasons (which mean nothing for true purists!).

```
        ;;HD takes one argument interpreted as a hexadecimal number (with optional sign and optional
        ;;fractional part) to (canonic) decimals. invalid input is returned as an empty value.
HD(X)   Q:$TR(X,"abcdefABCDEF",11111111111)'?.1"-".N.1".".N "" Q:"-."[X ""
        Q:$E(X)="-" "-"_$$HD($E(X,2,$L(X))
        Q:$L(X,".")=2 16**-$L(X,".",2)*$$HD($P(X,".",2))+$$HD($P(X,"."))
        Q $$HD1($TR(X,"0123456789abcdefABCDEF",
          $C(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,10,11,12,13,14,15)))
HD1(X)  Q:X="" 0 Q $$HD1($E(X,1,$L(X)-1))*16+$A(X,$L(X))
```

Figure 1. Hexadecimal conversion using only the QUIT command.

```
            ;;convert +$H-format date to ISO-format (honoring the Gregorian reform)
            ;;this is the main function, to be used from outside this context
DATE(H)     Q:$D(H)[0 $$DATE(+$H) Q:H="" "" Q $$D1(H+672411)
            ;convert modified +$H (counting from 1-JAN-0 AC) to ISO-format
            ;578101 is 15-OCT-1582, when the Gregorian reform took effect
            ;Russians replace that by 700579 (14-FEB-1918)
            ;do not accept values before 1/01/01
D1(H)       Q $S(H<366:"",H<578101:$$J(H),1:$$G(H))
            ;convert modified +$H-format to Gregorian calender format (ISO)
G(H)        Q $$ISO($$GY(H),$$GM(H),$$GD(H)
            ;convert modified +$H-format to Julian calender format (ISO)
J(H)        Q $$ISO($$JY(H),$$JM(H),$$JD(H))
            ;assemble date parts to ISO format
ISO(Y,M,D)  Q Y_"/"_$E(0,M<10)_M_"/"_$E(0,D<10)_"D"
            ;return the year of H (modified +$H format) in Gregorian calendar
GY(H)       Q H-1\146097*400+(H-1#146097\36524*100)+(H-1#146097#36524\1461*4)+(H-1#146097#36524#1461\365)
            ;return the year of H (modified +$H format) in Julian calendar
JY(H)       Q:H>3285 H+1\1461*4+(H+1#1461\365) Q H-1\365 ;(year 4 and 8 are **no** leap years)
            ;return the month of H (modified +$H format) in Gregorian calendar
GM(H)       Q $$GM0(H,$$GY(H))
GM0(H,Y)    Q $$XM($$GD2(H,Y),$$GL(Y))
            ;return the length of months before H (modified +$H format) in Julian calendar
JM(H)       Q $$JM0(H,$$JY(H))
JM0(H,Y)    Q $$XM($$JD2(H,Y),$$JL(Y))
            ;return the length of months before H (modified +$H format) in Gregorian calendar
GD(H)       Q $$GD0(H,$$GY(H))
GD0(H,Y)    Q $$GD1($$GD2(H,Y),Y)
GD1(D,Y)    Q D-$$XD(D,$$GL(Y))
GD2(H,Y)    Q H-(Y*365+(Y-1\4)-(Y-1\100)+(Y-1\400))
            ;return the length of months before H (modified +$H format) in Julian calendar
JD(H)       Q $$JD0(H,$$JY(H))
JD0(H,Y)    Q $$JD1($$JD2(H,Y),Y)
JD1(D,Y)    Q D-$$XD(D,$$JL(Y))
JD2(H,Y)    Q H-(Y*365)-$S(H>3285:Y-1\4-2,1:0)
            ;return whether Y is a leap year (in Julian calendar)
JL(Y)       Q:Y<12 0 Q Y#4=0
            ;return whether Y is a leap year (in Gregorian calendar)
GL(Y)       Q:Y#4 0 Q:Y#100 1 Q Y#400=0
            ;return combined length of previous months
XD(D,LP)    Q:D<32 0
+1          Q:LP $S(D<61:31,D<92:60,D<122:91,D<153:121,D<183:152,D<214:182,D<245:213,D<275:244,D<306:274,
            D<336:305,1:335)
+2          Q $S(D<60:31,D<91:59,D<121:90,D<152:120,D<182:151,D<213:181,D<244:212,D<274:243,D<305:273,
            D<335:304,1:334)
+3          ;return combined length of previous months
XM(D,LP)    Q:D<32 1
            Q:LP $S(D<61:2,D<92:3,D<122:4,D<153:5,D<183:6,D<214:7,D<245:8,D<275:9,D<306:10,D<336:11,1:12)
            Q $S(D<60:2,D<91:3,D<121:4,D<152:5,D<182:6,D<213:7,D<244:8,D<274:9,D<305:10,D<335:11,1:12)
```

Figure 2. $HOROLOG conversion using only QUIT command.

Example of use:

```
> Write $$HD("-CAFE.cafe")
-51966.792938232421875
```

Figure 2 is a function to convert $H formats to ISO date format (YYYY/MM/DD). Typical solutions do not care for nonpositive values, or worse, they work properly only for dates in the twentieth century. The functional approach presented here gives an easy extension to all $H values. Negative values are accepted, as long as no day before 1/01/01 is designated by that number. Dates before 1/01/01 are avoided, as some reckon with a year zero (astronomers) and others don't (historians). Naturally the switch

from Julian to Gregorian on 15-Oct-1582 is properly taken care of. If in your part of the world the switch from Julian to Gregorian has taken place at a different time, just replace the number 578101 (15-Oct-1582) in line D1 with the proper value. The best known exception is Russia. She switched as late as 14-Feb-1918 (use 700579) from *old style* to *new style*.

Examples of use:

```
> Write $$DATE(-672045),!,
$$DATE(-94311),!,
$$DATE(-94310),!,
$$DATE(55555)
1/01/01
1582/10/04
1582/10/15
1993/02/07
```

Don't wait for $MIRACLE if your M has $$extrinsics and QUIT. It is almost all you ever need for programming. The other thing is coffee. QUIT. ❖

Winfried Gerum is president of Winner Software, a company dedicated to serving the M community with tools, consulting, and related services. His approach to M code has been a stimulant for the international M community.

In the November 1992 issue of *MUMPS Computing*, an inadvertant error occurred in the *Tips 'n' Tricks* column, despite every attempt to prevent mistakes. The editors sincerely regret the mishap. Here is the columnist's correction for page 26, column 2 "Guess What the Current Century Is."

The "Current Century" (in Gregorian calendar) is always given by

```
$H+672411\36524.25
```

Some of the few programmers acknowledging the nonconstancy of "Current Century" usually write the equivalent of

```
$H>58073+19
```

This is better than just writing 19, but not safe forever. Be careful: the formula for "Current Century" cannot be modified to calculate the "current year":

```
$H+672411\365.2425
```

This gives an approximation to the "current year." It may fail in the first or last day of some years; the next two dates of failure are 1-JAN-1996 ($H of 56613 => 1995) and 31-DEC-2036 ($H of 71588 => 2037). The fact that it never fails at the turn of the century makes the above century formula valid. ❖