Naked Reference— Beauty Queen or Porn Star?

by Winfried Gerum

Winfried Gerum

Programming languages are in some sense universal, natural languages are not. So many phrases or even single words may escape an easy translation. Can you imagine a German translation of the slogan "Follow the Leader" (motto of a well known M implementor)? Translated as *Folge dem Führer*, it probably gives the reader a very different feeling from the English version. In translating the term *naked reference*, authors of German textbooks encounter a similar problem: The word *nackt* is the dictionary entry for naked/nude. But the connotations for German readers, except for medical personnel, are such that everybody seems to avoid that word in textbooks or in articles.

Naked reference is a powerful mechanism in M programming. As with so many powerful things there are dangers using it. With the appropriate care these dangers can be mastered.

The standard says about naked reference:

> An abbreviated form of subscripted gvn is permitted, called the *naked reference*, in which the prefix (i.e., ^) is present but the name and an initial (possibly empty) sequence of subscripts is absent but implied by the value of the *naked indicator*.

The naked indicator is the last actual global reference in the course of code evaluation, stripped of its last subscript. If the last global reference is to an unsubscripted global, the naked indicator becomes undefined and the use of the naked reference is erroneous.

The effects of various situations on the naked indicator are scattered around the whole text of the standard. Nevertheless it is quite easy to remember what affects the naked reference: Each actual access to a global variable affects the naked indicator. So all you have to analyze is the sequence of evaluations of M code.

Usually that is quite simple, as the sequence of evaluations is in most cases simply from left to right. But it does not always come in the most intuitive way:

```
SET ^GLOBAL(SUB1,SUB2)=^GLOBAL
(SUB1,SUB2)+1
```

Here, first the name on the left side of the SET command is evaluated, but that does not affect the naked indicator, unless SUB1 or SUB2 are referencing a global. Then the right-hand side is evaluated, setting the naked indicator to `^GLOBAL(SUB1,` permitting a rewrite of the whole line to

```
SET ^(SUB2)=^GLOBAL(SUB1,SUB2)+1
```

The latter code is in many cases preferable to the former code: If SUB1 is an expression, you have the choice of repeating this expression on both sides of the SET or of introducing a variable. So for the advanced programmer, efficiency *and* readability are enhanced by using the latter code.

The next piece of code

```
IF$O(^($O(^GLOB1(SUB1,SUB2,"")
)))=""
```

is completely unambiguous, but not so straightforward to read. The condition tests whether on the level `^GLOB1(SUB1,SUB2,. . .)` there is at most one subscript with `$DATA()>0`. The alternate code

```
S X=$O(^GLOB1(SUB1,SUB2,"")) IF
'$O(^GLOB1(SUB1,SUB2,X))=""
```

may be considered more readable. The cost is a variable X avoided in the shorthand.

There are several circumstances under which a global might not be referenced at all. The most obvious is after an IF or ELSE:

```
  IF COND1 SET X=^GLOB1(SUB1)
  ELSE SET X="DEFAULT"
  SET Y=^(SUB1,X)
```

If COND1 is false, the code on the right side of the IF is not evaluated, and the naked reference is the same as before the IF. Likewise after a FOR:

```
SET X=""
FOR SET X=$ORDER(A(X)) QUIT:
X ="" SET ^GLOB1(X) =A(X)
SET LAST=^(X)
```

In this case, IF $DATA(A)<10, the FOR loop never goes beyond the QUIT, and the naked indicator is not changed by the FOR loop.

```
SET X=$S(C0:A,C1:^G1(Y),C2:^G2
(Y),^C3(Y):"EXC",1:"DEF")
```

In the case of this $SELECT() there may be zero to one accesses to globals, leaving the naked indicator either

```
  unchanged
  ^G1(
  ^G2(
  ^C3(
```

as the $SELECT() evaluates only one of the following combination of exprs

```
  C0,A
  C0,C1,^G1(Y)
  C0,C1,C2,^G2(Y)
  C0,C1,C2,^C3(Y),"EXC"
  C0,C1,C2,^C3(Y),"DEF"
```

The recently introduced second argument $GET() is not equivalent to code using the very similar-looking $SELECT():

```
SET X=$GET(^VALUE(A),^DEFAULT(A))
SET X=$SELECT($D(^VALUE(A))#2:^
(A),1:^DEFAULT(A))
```

After the $GET() the value of the naked indicator is always ^DEFAULT(. After the $SELECT() the value is either ^VALUE(or ^DEFAULT(.

$SELECT() skips exprs not needed for computing its value, whereas $GET() evaluates all arguments. This has been designed exactly to simplify the effect on the naked indicator, but it may be less efficient in some cases. And there are dangers if you want to rewrite code with $SELECT() by $GET(), precisely because of the effect on the naked indicator and because with $GET() the second argument has to be *evaluatable* in any case.

With SET $PIECE and SET $EXTRACT there is a subtle booby trap:

```
SET $PIECE( ^GLOB1(SUB),";",FR,
TO)=EXPR
```

and

```
SET $EXTRACT( ^GLOB1(SUB),FR,TO)
=EXPR
```

In both cases, if FR>TO there is *no* access to the global ^GLOB1() and the naked indicator is left unchanged! This may apply very rarely, which does not make it less dangerous.

Usually programmers speak about LOCK*ing* globals. But you should be aware that from the viewpoint of the language, *namespaces*, but not globals, are locked. So if you specify a global variable name as a LOCK argument, there is no access to the global and hence the naked indicator is not affected:

```
  LOCK ^PATDAT( ^NEXTPAT(NOW))
  SET ^(X) =NEWVALUE
```

The naked reference of the SET refers to ^NEXTPAT(X) and not to ^PATDAT(X).

In some respects the naked reference (naked indicator) is similar to the $TEST special variable. Its value does not depend on the syntax, but on the sequence of evaluation. Likewise in $TEST you should always use the naked reference with the utmost care.

In maintenance of software you have to be especially careful. For example, when changing

```
SET ^("TOTAL")=^INVOICE(X,"TOT
AL")*VAT
```

to

```
SET ^("TOTAL")=^INVOICE(X,"TOT
AL")*$$VAT(INVDAT)
```

there may be a problem: If $$VAT() accesses a global variable it messes around with the naked indicator!

There is one circumstance for which implementors had different ideas about the value of the naked indicator: the $QUERY() function. This accepts array references as its argument and returns the next storage reference in the tree representing the array. So the argument of $QUERY and the returned value may have a different number of subscripts. Some see the argument as the last accessed global reference, some see the value returned as the last accessed reference. So using naked reference after a $QUERY() function may not be portable. Sentiment in the MUMPS Development Committee is to make the naked indicator explicitly undefined after $QUERY.

Using naked reference with one subscript does not change the value of the naked indicator. But using it with two (or more) subscripts *does* change the naked indicator: Writing ^(SUBN) after a previous ^GLOB(SUB1,SUB2) is just shorthand for ^GLOB(SUB1,SUBN) which results in the same naked indicator. But writing ^(SUBN1,SUBN2) as shorthand for ^GLOB(SUB1,SUBN1,SUBN2) changes the naked indicator to

`^GLOB1(SUB1,SUBN1`. This surprises some beginners who tend to think that using the naked reference never changes the naked indicator.

There are some situations where this makes an extremely nice and efficient tool. In the next issue of *M Computing*, the *Tips 'n' Tricks* column on data compression will show M code using naked reference quite extensively—clearly demonstrating naked reference as a beauty queen.

In conclusion, I have to admit that the question posed in the title—beauty queen or porn star?—cannot be clearly decided either way. I have seen many nasty programming errors, especially after doing *small* changes in maintenance, result from the use of naked reference. Often it is quite hard to detect these errors, as the consequences may show up in entirely different situations. **M**

Winfried Gerum is president of Winner Software, GmbH, in Erlangen, Germany. His column appears regularly in *M Computing*. A similar article appeared in the *MUG-E Newsletter* earlier this year.

The Year of The Standard

What's Happening to Our M Standard? Who Decides?

How will validation and internationalization affect you in your everyday work?

A subscription to the MUMPS Development Committee mailings is the definitive source for all relevant M standard information on

X11.1, GKS, TCP, OMI, and WINDOWING API —

Everything everyone in the M community is talking about today.

Each mailing contains STIMULATING, INVALUABLE MDC RESOURCES to help you understand the latest work of the MDC, its five major subcommittees, and more than forty task groups. A YEAR'S SUBSCRIPTION BRINGS YOU AT LEAST SIX MAILINGS — two for each MDC meeting.

All this will be yours for ONLY $130.00 (plus shipping and handling) — a FULL YEAR of information on M Technology and the M Standard. ORDER YOUR SET ON PAGE 53.

Don't be left out of the knowledge loop! These are VALUABLE publications to help you TRACK EACH DEVELOPMENT of the M STANDARD.

If you've been following developments about the VA FileMan version 20.0 in *M Computing,* you know you've got to have it! Buy it from your Association TODAY! Only $100 buys FileMan 20.0 and all documentation. Call MTA today — specify 5 1/4" or 3 1/2" floppy (HD) disk, or magnetic tape. Just need the manuals? Only $60. Phone: 301-431-4070. Have your MasterCard or VISA card ready!

Programmers/Analysts

COMSYS, an INC. 500 Co., has IMMEDIATE openings for programmers and analysts with 1+ yrs. of MUMPS full life cycle development exp. for the Washington, D.C., area. Responsibilities include design, development, and programming of a health care information system for multi-hospital environment.

Qualified candidates please call 301-921-3659 or 800-926-6797 and SEND/FAX (301-921-3700) a detailed resume including HW/SW and salary req. to:

COMSYS
Attn: B. Bawcombe
Dept. MUG1193
P.O. Box 7947
Gaithersburg, MD 20898-7947
(EOE)