# Do You Know All About $DATA?

*by Winfried Gerum*

*Winfried Gerum*

Novices quickly learn how useful MUMPS is, but experts, too, find surprising new uses of MUMPS syntax. I recently went into code reading

```
Q:$S($D(X):1,1:0)
I '$D(X)#10
I $D(X)\10
I $D(X)\2=0
```

and wondered what other people may or may not do with $DATA. To begin, let's refer to what the standard says about $DATA: 2.2.73 $DATA states $D[ATA] (glvn). This form returns a nonnegative integer, which is a characterization of the glvn. The value of the integer is $p+d$, where:

$d$ = 1 if the glvn has a defined value, i.e., the NAME-TABLE entry for the name of the glvn exists, and the subscript tuple of the glvn has a corresponding entry in the associated DATA-CELL; otherwise d=0.

$p$ = 10 if the variable has descendants; i.e., there exists at least one tuple in the glvn's DATA-CELL that satisfies the following conditions:

a. The degree of the tuple is greater than the degree of the glvn, and

b. The first N arguments of the tuple are equal to the corresponding subscripts of the glvn where N is the number of subscripts in the glvn.

If no NAME-TABLE entry for the glvn exists, or no such tuple exists in the associated DATA-CELL, then $p$=0.

The $DATA function is useful in two situations. First, there may be uncertainty as to whether some data are available. A subroutine or a previous routine may have altered the local variables. Other jobs may have changed the value independently. Second, an UNDEFINED state of a variable is an acceptable state of a MUMPS variable. By convention, therefore, the lack of definition itself may carry information: a flag may be implemented by a variable being either undefined or having some arbitrary (defined) value. If one has a variable that assumes one of its possible variables frequently, then there is a lot of space to be saved, if that value is by convention implied by an UNDEFINED state. In the first situation, global variables are proper, but usually indicate bad style with local variables.

$DATA actually gives two bits of information, i.e., four different states denoted 0,1, 10, 11. There are 2**4 = 16 distinct subsets of its values. In many situations, only one of these two bits are of interest. As long as there actually is a variable, either subscripted or unsubscripted, it is a simple matter. If one uses vintage code, it is better to take no risks and check for specifics. Three MUMPS vendors' programs that were part of their implementations contained no subsets coded thusly:

```
IF $D(X)=1!($D(X)=11)
```

Since MUMPS programmers like to be concise, I found the following solutions for the routines I analyzed, with the last one being the shortest solution:

```
$D(X)#10
$D(X)#10=1
$D(X)#2
```

A full analysis shows the following usage

| | | | | | Vendor 1 | Vendor 2 | Vendor 3 |
|---|---|---|---|---|---|---|---|
| Number of routines | | | | | 269 | 91 | 220 |
| Number of $DATA calls | | | | | 918 | 193 | 208 |
| Of these $D(^...) | | | | | 294 | 61 | 208 |
| | 0 | 1 | 10 | 11 | | | |
| Subset# | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | – | – | – |
| 1 | 1 | 0 | 0 | 0 | 319 | 67 | 146 |
| 2 | 0 | 1 | 0 | 0 | 2 | – | 1 |
| 3 | 1 | 1 | 0 | 0 | 18 | 1 | 3 |
| 4 | 0 | 0 | 1 | 0 | – | – | 1 |
| 5 | 1 | 0 | 1 | 0 | 11 | 4 | 3 |
| 6 | 0 | 1 | 1 | 0 | – | – | – |
| 7 | 1 | 1 | 1 | 0 | – | – | 2 |
| 8 | 0 | 0 | 0 | 1 | – | – | – |
| 9 | 1 | 0 | 0 | 1 | – | – | – |
| 10 | 0 | 1 | 0 | 1 | 47 | – | 57 |
| 11 | 1 | 1 | 0 | 1 | – | – | 4 |
| 12 | 0 | 0 | 1 | 1 | 8 | – | 5 |
| 13 | 1 | 0 | 1 | 1 | 1 | – | – |
| 14 | 0 | 1 | 1 | 1 | (512) | (121) | 1+(310) |
| 15 | 1 | 1 | 1 | 1 | – | – | – |

All three vendors supply about the same number of routines. Two of them, however, supply part of the routines in compiled form only. There are surprising differences remaining when the numbers are corrected for the different numbers of routines in each set. Vendor Two uses far less $DATAs than its competitors and its routines frequently use functions and procedures. Another explanation for differences in the frequency of $DATA calls might be the newly introduced $GET() function. Perhaps $GET() replaces a frequently used construction with $DATA(). I checked for that effect:

| | Vendor 1 | Vendor 2 | Vendor 3 |
|---|---|---|---|
| number of $GET calls | 81 | 199 | 876 |
| number $S($D(..):..) | 29 | 7 | 25 |

There does seem to be a tradeoff between $DATA() and $GET() with vendors One and Two. Analysis of Vendor Three revealed that some people use $G() more often than they would use $$($D():(),1""). All of these packages have roughly one-third of the $DATAs on globals, and the hypothesis that "good" code uses fewer $DATAs on locals has no support in the analyzed material.

The most frequent use of $DATA is without an explicit reference to its four possible values, which looks like subset 15. But $DATA is in almost all cases part of a boolean expression (tvexpr). Because the implied boolean interpretation selects implicitly subset 14, all uses of $DATA not explicitly referring to another subset counted in subset 14. There was only one explicit reference to this subset in all of the routines scrutinized ($D(X)>0). Not surprisingly, the complement subset 1 is the second-most frequent one. Despite the simple coding '$D(X), one can find alternatives such as:

```
$D(X)'>0
$D(X)<1
'$D(X)#10
'$D(X)#2
```

for subset 1.

Of practical importance are subsets 5 and 10; subset 10 is the complement (negation) of subset 3. Both essentially test for the same situation: If $D(X) happens to be 10, then code such as

```
IF $D(X) W X
```

or

```
WRITE $S($D(X):X,1:"")
```

will result in an error and it is better to write:

```
IF $D(X)#2 WRITE X
```

or

```
WRITE $S($D(X)#2:X,1:"")
```

In more than ten years' MUMPS experience, I have seen only one application crash with <UNDEF> of variable ^GLOBAL(VAR) in this situation:

```
WRITE $S($D(^GLOBAL
(VAR))#2:^(VAR),1:"")
```

Another job actually killed the global between $DATA and the subsequent reference. It is good to avoid the situation altogether with the $GET function.

Since programmers are very familiar with subset 10, they begin thinking there when they need a solution to compute subset 5. In some cases, there was

```
IF '($D(X)#2) READ !,"X NEED A
VALUE",X
```

the frontrunner is

```
IF $D(X)#2=0...
```

A shorter solution not found in the routines I analyzed was

```
IF $D(X)[0
```

This may be because programmers think of $DATA as having a numeric value and the "contains operator" is regarded as a string operator. Therefore, they rarely, if ever, write this combination. In the same way, it is unlikely that IF 10[$D(X) will be present instead of IF $D(X)<11.

MUMPS has no data types; anything is string and is interpreted as numeric or boolean as needed. Any operator accepts nearly any data (except perhaps the numeric interpretation of a legal string may be an illegal number, as in X_"1E9000" and X+"1E9000").

The following table shows the shortest way to test for each subset. Nowhere is there a shorter solution. Some have alternatives of equal length and thus the most natural solution has been selected here.

Only two subsets, 6 and 9, again complements, cannot be computed with a single operation. This does not present a problem, because these subsets are not relevant in everyday problems. If a value is interpreted as a boolean value (truth value), however, there is no need to look for zero/one results. Then we just care for zero/non-zero. In the case of subset 14, the boolean process does the job by itself. In subsets 11 and 13, the negated comparison may be replaced by a subtraction. Subset 6 can be achieved in a single operation that does not resemble the "correct" solution.

There is also an elegant solution to map the $DATA values into four consecutive integers. $D(X)#4 maps (0,1,10,11) into (0,1,2,3), such that

```
W $P("zero^one^ten^eleven","^",
$D(X)#4+1)
```

verbalizes the value of $DATA(X).

$DATA and $GET are useful but they should not be used to "cure" bad code. Code should not produce unexpected UNDEFINED values (nor unexpected DEFINED values either). Extensive use of modern syntax with extrinsic functions, procedures and NEW is of great help. ❖

This column is based on a column that was first published in the **MUG-Europe Newsletter**, vol. IX, no. 1 (1992).

Winfried Gerum is president of Winner Software, which he started in 1991. The company is in Erlangen, Germany, and is dedicated to providing the M community with tools, consulting, and related services. Gerum has been active in MUMPS for many years, writing articles, contributing proposals to MDCC-Europe, and helping organizing last year's MUG-Europe conference in Nuremberg.

*Subset functions of $DATA*

| subset # | 0 | 1 | 10 | 11 | MUMPS Code | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | FALSE (no value accepted) |
| 1 | 1 | 0 | 0 | 0 | '$D(X) | |
| 2 | 0 | 1 | 0 | 0 | $D(X)=1 | |
| 3 | 1 | 1 | 0 | 0 | $D(X)<2 | |
| 4 | 0 | 0 | 1 | 0 | $D(X)=10 | |
| 5 | 1 | 0 | 1 | 0 | $D(X)[0 | |
| 6 | 0 | 1 | 1 | 0 | $D(X)#3=1 | or, as tvexpr: $D(X)#11 |
| 7 | 1 | 1 | 1 | 0 | $D(X)<11 | |
| 8 | 0 | 0 | 0 | 1 | $D(X)=11 | |
| 9 | 1 | 0 | 0 | 1 | $D(X)-1[1 | |
| 10 | 0 | 1 | 0 | 1 | $D(X)#2 | |
| 11 | 1 | 1 | 0 | 1 | $D(X)'=10 | or, as tvexpr: $D(X)-10 |
| 12 | 0 | 0 | 1 | 1 | $D(X)>1 | |
| 13 | 1 | 0 | 1 | 1 | $D(X)'=1 | or, as tvexpr: $D(X)-1 |
| 14 | 0 | 1 | 1 | 1 | $D(X)>0 | or, as tvexpr: $D(X) |
| 15 | 1 | 1 | 1 | 1 | 1 | TRUE (all values accepted) |