# 1. Identification of the proposed change

## 1.1. Title: $MUMPS Function

## 1.2. MDC Proposer and Sponsor

Alan Frank
35 Gardner St.
Arlington, MA 02174
Phone: 781 647-4884x3280 (at Dynamic)
Fax: 781 648-2824
alf@matchups.com

SC15, Programming Structures
Chair: Wally Fort
VAISC6 - San Francisco
301 Howard St. #600
San Francisco, CA 94105
fort.w@forum.va.gov; 415-744-7520

## 1.3. Motion: That the MDC approve this document as a type A, superseding X11/SC15/TG9/97-4.

## 1.4. History:

| | |
|---|---|
| June, 1998 | X11/SC15/98-8 This document, with no substantive changes. |
| March, 1998 | No new document because previous meeting's minutes had not been published. |
| September, 1997 | X11/SC15/TG9/97-4. No changes outside of sections 1.4 and 7. Passed as type A by SC15, 17:2:5. |
| January, 1997 | X11/SC15/TG9/97-1 In order to enable this proposal to advance in the absence of a detailed list of syntax errors, a change was made specifying how the implementation should proceed in this event. Passed by TG9, 8:0:0 and by SC15, 17:4:3 |
| 1996 | SC15 did not approve a document providing specific codes for syntax errors |
| September, 1995 | X11/SC15/95-108 Identical to the previous one, approved as type A by subcommittee |
| June, 1995 | X11/SC15/95-28, improves language regarding implementation extensions, approved as type B 13:5:6 by subcomittee. |
| January, 1995 | X11/SC15/94-10/, reflects amendments, fills in section 4.3. Remanded to taskgroup by subcommittee. |
| June, 1994 | X11/SC15/TG10/94-6, as amended in task group, passed as revised subcommittee type B, 20:0:5. |
| January, 1994 | X11/SC15/TG10/94-1 Task group recommended making similar to $ECODE |
| October, 1993 | X11/SC15/TG10/93-3 Task group recommended to make return value a list. |
| February, 1993 | SC 15 remanded X11/SC15/93-16 to task group |
| 1987 | Passed as SC1 Type B |

## 1.5. Dependencies

This proposal depends on a proposal to list syntactical error codes.

# 2. Justification of proposed change

## 2.1. Needs

Although there are many advantages to program development in the interpreted nature of MUMPS, an important disadvantage is that syntax errors in programs may not become apparent until the programs run. The $MUMPS function provides a means for editors or routine generators to warn the programmer of code which may cause a run-time error.

## 2.2. Existing practice:

Code is not syntax-checked, or application developers write their own parsers, which require updating as the language changes.

# 3. Description of the proposed change

## 3.1. General description

A function $M[UMPS] is created which takes one argument and returns a value indicating whether the argument is a legal MUMPS line. If it isn't, additional information may be returned.

- If the argument is a legal line of Standard MUMPS code, the function returns the string "0".
- If the argument will produce a syntax error if executed, the function returns a value whose first " ; " piece is a positive integer.
- Otherwise (the argument involves an extension to Standard MUMPS), the function may return either of the above.

Implementors are urged to return the approximate location of the error in the first " ; " piece (or a value greater than the length of the string), but this is *not* required and the MUMPS programmer should not make any assumptions based on the magnitude of the function result.

If the first " ; " piece is positive, the second " ; " piece contains a code identifying the nature of the syntactic error. This code will start with "M" or "S"[1] if the error is in a standard construct or "Z" if the error is in the use of an implementor extension. If no standard error code has been defined which describes the error, code S0 shall be used.

The third " ; " piece is available for narrative description of the error. Additional " ; " pieces are reserved. Multiple errors may be reported by means of a comma-delimited list. Note that both semicolons are required, to facilitate "contains" checking (e.g., `SET R=$MUMPS(CODE) QUIT:'R IF R["S10;"...`)

## 3.2. Annotated examples of use

Note: legal results are listed in decreasing order of preferability and are not a complete list.

Argument: `"TAG S X=5"` Only legal result: `0`

Argument: `" S X=,B=22"`

Legal results:
```
5;S1;Unexpected comma
5;S0;
999;S1;
8;S1;
5;S1;,12;S10;
76;S0;
```

Illegal results:
```
-3 FOO
5;S1
0
5
76 TROMBONES
Unexpected comma
```

Argument: `" W $L(A,B,C),$HOORLOG"`

Legal results:
```
10;S10;,20;S20;
10;S10;Too many arguments
20;S20;Unknown svn
```

Argument: `" ZB 1"` Legal results:

---

[1] A separate proposal will reserve the letter S for this purpose and provide an initial standard list of such codes. There will be a code (S0) to indicate that the implementor doesn't care to report what the problem is. The examples below use arbitrary codes

```
Depending on the implementation,          1;S5;
                                          0
                                          1001;;NIGHTS
                    Illegal results:      1;S42;Undefined variable

Argument: " O 10:ZMODE=""APPEDN""""   Legal results: 7;Z20;Unknown mode
or, depending on the implementation,      0
                                          7;S0;
```

## 3.3. Formalization:

Note that section references are to version 9 of the RMDS.

Add a new section 7.1.5.9.5 SMUMPS and appropriate cross-references:

**7.1.5.9.5 SMUMPS**

SM[UMPS] ( expr )

Let *s* be the value of expr with eol appended.

a. If *s* matches the syntactic definition of a line, without any implementor ("Z") extensions, the function returns the number 0.

b. If *s* matches the syntactic definition of a line only because of syntactic extensions to the language available in the current implementation, the function either returns the number 0 *or* provides a return value as in case c.

c. If *s* does not match the syntactic definition of a line, the function returns a value of the form

```
$$ mumpsreturn
```

```
mumpsreturn ::= intlit ; ecode ; [ noncommasemi ... ]
```

noncommasemi ::= any of the characters in graphic except the comma character and the semicolon character

In each mumpsreturn, the ecode must be one which actually describes an erroneous condition in *s*. If no standard ecode describes an erroneous condition in *s*, an ecode of S0 shall be used. All nonpositive values of intlit are reserved for future enhancement of the standard.

# 4. Implementation effects

## 4.1. Impact on existing user practices and investments

Will enable more errors to be caught at development time, rather than run time. Will not affect existing code.

## 4.2. Impact on existing vendor practices and investments

It is expected that most vendors already have internal entry points which provide, at least, the Boolean functionality required by this proposal. Consequently, this should be implementable with minimal work.

## 4.3. Techniques and costs for compliance verification

It's not possible to thoroughly test this function, but you can pass all the lines in all the routines on a system and make sure you get back 0. Then pick each possible syntax error and create a string with that error and make sure it's detected. Also create strings with multiple errors and see that they are properly checked. Finally, make sure implementation- specific code is handled in one of the two legal ways.

## 4.4. Legal considerations:

It should be stated somewhere in the document that values returned by this function may change over time due to further additions to the language. For example, if this function were implemented on an older system (that is, lacking two-argument $ORDER), `$MUMPS(" S X=$O(@R,-1)")` would give an error return, whereas if it were on a system with the two-argument $ORDER, it would not give an error return.

# 5. Closely related standards activities

## 5.1. Other X11 proposals under consideration

X11/SC15/TG9/97-5: Standard routineparameter for syntax checking

## 5.2. Other related standards efforts: none

## 5.3. Recommendations for coordinating liaison: none

# 6. Associated Documents: none

# 7. Issues, Pros and Cons, and Discussion:

It was confirmed in October, 1993 by the task group that the function should be required to let you know whether the code is okay on the current system (and not whether it's "standard") for three reasons: it's easier for vendors to implement, it's more useful, and it's harder to do with a MUMPS function.

The following Pros and Cons were raised at the September, 1997 meeting of Subcommittee 15:

**Pros:**

1. Addresses perceived need [8]
2. Has been implemented [8]
3. Foundation for standard routineparams [6]

**Con:**

1. Unpredictable results [2]

In syntax errors, it is often difficult to determine exactly what is wrong (e.g., too many or too few quotes), so that different implementations may return different errors. Also, this committee has indicated a lack of interest in publishing a list of standard syntax error codes.

The following Pros and Cons were raised at the March, 1997 meeting of Subcommittee 15:

**Pros:**

1. Has been implemented [5]
2. Needed now [6]

**Cons:**

1. Should be library function [0]

This is a standard con, but no voter cited it.

2. Belongs to development environment and not language [5]

There is no X11 standard for a development environment. Also, for CASE tools written in MUMPS, this function will be useful.

The following Pros and Cons were raised at the January, 1995 meeting of Subcommittee 15:

**Pros:**

1. Functionality has been implemented
2. Will improve software quality

**Cons:**

1. Not well-defined & therefore not portable

This depends on what you mean by "portable." If you mean that it will always produce the same result on every implementation, then like $SYSTEM it is not "portable." However, if you mean that the same syntactic construct will always produce the answer to the same question, formatted in the same way, then it is portable.

2. Other means available

Some implementations may have non-standard mechanisms for this functionality, but the only ones the document editor knows about apply to a routine, rather than a single line; besides, such mechanisms are not provided by all implementations, nor in a consistent fashion.

There are also public-domain utilities for syntax checking. Such utilities, unlike the proposed function, would not automatically reflect the language as implemented, including vendor extensions or future revisions of the standard.

3. Not all errors are reportable by *ecode*

Another proposal is being brought forward to allow all syntax errors to be identified by ecodes. The proposer does not consider this Con to be relevant to this proposal.