MUMPS Language Issues

Thomas C. Salander
The Connections Group
3629 Kimble Road
Baltimore, MD 21218 301-889-0447

Datatypes: Strong, Weak and Imaginary

"MUMPS is typeless." "We only have one data type." "Everything is a string."

Depending on your point of view, these are all true statements. MUMPS is called typeless because there are no type declarations. The explicit data type for data storage is the variable length character string. Any variable can be assigned any literal value.

But viewing MUMPS as "typeless" may lead to problems. In fact, MUMPS has many data types, although most of them are implicit. Strongly typed languages, such as Ada, Algol, and FORTRAN, require a variable to be declared with an explicit data type before execution. The primary justification for this static typing is the reduction of errors within modules of code. At the other end are the dynamically typed languages such as Bliss, MUMPS and REXX. Dynamic typing provides the programmer a much greater degree of freedom in both program design and implementation.

MUMPS is a high level language with a less formal, programmer controlled type management. This flexibility requires a more sophisticated automation of type interpretation. MUMPS is context sensitive. That is, a given character, operator, or variable will be interpreted differently depending on the context in which it is being used. For the determination of data type, MUMPS will take one of two approaches. Sometimes MUMPS will take the whole data value and treat it as if it were the "correct" (meaning expected) data type. In other contexts, MUMPS will use as much of the data as makes sense (starting from the left) for the expected data type.

This second method is called type coercion. Type coercion is the process MUMPS uses to make the transformation from one data type to another. When used to discuss people, coercion implies getting someone to do something that they might not normally do (or to do something against their better judgment). In MUMPS, the coercion is less sinister (unless you are an advocate of strong, explicit data typing). Generally, coercion will occur whenever a numeric, integer, or truth value is expected. We also can refer to type coercion as type conversion.

All operators, most functions, and a few commands will coerce the correct data type. The rule is, if the expected data type is a siring, numeric, integer, or truth value, coercion can be used. If the expected data type is anything else, the actual data must be complete and in the correct format. The return values are always in the correct format for the indicated data type.

Understanding the different MUMPS data types is not just an academic exercise. When we develop MUMPS routines, we usually use expressions as the arguments of functions. Less frequently, but just as importantly, we use expressions as the arguments of commands. We also build expressions from other expressions (the onion approach to coding). Implicit in this is the expectation that we know what data type we need in a given context and what data type we will get back by each operation. (And if we don't, then we better not go anywhere NEAR indirection!)

What follows is a discussion of MUMPS data types: explicit, implicit, and imaginary. I have listed where the particular data type is expected (or coerced), and where it is returned. Note that some functions will take more than one data type in the same location.

String

Anyone who has programmed in MUMPS, read an article on the language, or talked with any of the MUMPS zealots, has heard that the string is the only MUMPS data type. A string is any collection of characters, 0 or more characters in length. (For portability, the standard restricts the length to a maximum of 255 and the characters must be from the ASCII character set.) That's it. It can be anything. It is this free formed anarchy that can send shivers down the spine of a strong data typing advocate.

The string is the result of evaluation of an expression. It is from the string that MUMPS derives most of the other data types. The string and the data types that can be derived from a string are often called simple data types. In MUMPS, the distinction is with the address data type (discussed below starting with names).

Format: any collection of characters; variable length

Coercion: all characters are taken without modification

Examples: the list below contains examples of strings. Note that the quotation marks are included as you would use the string in an assignment operation (`set x="TEST"`). The quotes are not part of the string.

EXAMPLES:

"12345"

"49.95"

"-2"

"25Kate"

"MUMPS Users"

"0.30-"

"+-+-5-"

"+18-6"

"TEST"

"-TEST"

"3.20E5"

"-3.20E5"

"12345"

"-3.20E-5"

"3.20Elenor"

"3.20e5"

"3.20E2.5"

"" (null)

INPUT TO:

function or operator	argument	comment
?		pattern match
[		contains
]		follows
=		equals; except for the first occurrence in the argument of the SET command, the equals is a truth-operator and is checking string identity.
_ (underscore)		concatenate
$ASCII	1st	
$EXTRACT	1st	
$FIND	1st, 2nd	looks for the 2nd argument within the first argument.
$JUSTIFY	1st	is viewed as a string only if there is no 3rd argument present.
$LENGTH	1st	
$PIECE	1st	the string from which a piece is to be returned.
$PIECE	2nd	the delimiter that identifies the pieces within the 1st argument.
$SELECT	2nd	This is actually the second piece (by colon) of every argument of the function.
$TRANSLATE	1st	the string to be translated
	2nd	the list of characters to be translated in the 1st argument.
	3rd	the characters that will replace the characters identified in the 2nd argument.
CLOSE command		the simple form, without parameters, may be an expression that is evaluated as a string.
OPEN command		the simple form, without parameters, may be an expression that is evaluated as a string.

READ command		will display any string used as an argument if the expression that produced the string is a literal.
USE command		the simple form, without parameters, may be an expression that is evaluated as a string.
WRITE command		will display any string resulting from an expression used as an argument.

Returned By:

function or operation	**comment**
_(underscore)	concatenation
$IO	special variable
$CHAR	
$EXTRACT	
$FNUMBER	while this function formats numbers, the results are usually NOT pure numeric values.
$GET	either the value of the variable identified in the argument, or a null
$JUSTIFY	
$PIECE	
$SELECT	returns the string associated with the first true expression.
$TRANSLATE	returns the 1st argument with all the characters identified in the second argument replaced by the corresponding characters in the second argument.

Numeric

Take a string and look at each character starting from the left. When you find a character that no longer makes sense as a number, stop. Everything before that is the numeric value of the string. Minus and plus signs (any number) may occur at the beginning and still count as numeric. They are not valid after any other numeric characters (that is, 012+345 is not a valid numeric). A decimal point can occur once, but not twice. Other than these exceptions, any non-numeric character will end the numeric interpretation of a string.

The exception to this description is exponentiation. Any number may be followed by an E (for exponential notation) that is followed by a plus or minus sign followed by one or more integers.

Once MUMPS finds that portion of the string that is the numeric value, MUMPS will do a final transformation into the canonic form of the number. A canonic number is the decimal value without leading or trailing decimal zeros and a leading sign only for negative numbers.

It is possible to use MUMPS code to determine the numeric value of any string, by using the unary operator plus ("+").

Example:	`set number=+string`
Format:	optional minus sign followed by one or more numeric characters (the first numeric is not a zero)
or	optional minus sign followed by zero or more numerics followed by a decimal point followed by one or more numerics (the last numeric is not a zero)
or	zero

Coercion:

1. scan the string from left to right until the first character that does not "make sense" as a number is found
2. truncate from that character to the end of the string
3. resolve multiple signs to a single sign
4. convert to decimal form
5. drop leading plus sign
6. drop leading minus sign on zero value
7. drop leading zeros
8. drop trailing fractional zeros

Examples:

string	numeric portion	canonic number
12345	12345	12345
49.95	49.95	49.95
-2	-2	-2
25Kate	25	25
-0.3	0.3	0.3
+-+-5-	+-+-5	5
+18-6	18	18
TEST	(note: null)	0
-TEST	-	0
3.20E5	3.20E5	320000
-3.20E5	'-3.20E5	-320000
-3.20E-5	-3.20E-5	-.0000032
3.20Elenor	3.20	3.2
3.20e5	3.20	3.2
3.20E2.5	3.20E2	320

Input To:

operator	argument	comment
+		addition
-		subtraction

*		multiplication
/		division
\		integer division
#		modulo
<		less than
>		greater than
$JUSTIFY	1st	The first argument is assumed to be numeric only when the third argument (number of decimal places for rounding) is present.
$FNUMBER	1st	The first argument is always assumed to be numeric.
HANG command		
timeouts		

RETURNED BY:

	function or operation
+	addition
-	subtraction
*	multiplication
/	division

Integer

Integers are always whole numbers. To coerce an integer, MUMPS starts with a numeric and then drops (truncates) the decimal and all digits that follow. No rounding takes place.

To use MUMPS code to determine the integer value of any string (and remember that this would include numbers), an integer division by 1 can be used.

Example: `set integer=string\1`
Format: optional minus sign followed by one or more numeric characters (the first numeric is not a zero)
or zero

COERCION:

1. coerce numeric value
2. delete decimal point and all fractional digits

EXAMPLES:

string	**canonic number**	**integer value**
12345	12345	12345

49.95	49.95	49
-2	-2	-2
25Kate	25	25
.30-	0.3	0
+-+-5-	5	5
+18-6	18	18
TEST	0	0
-TEST	0	0
3.20E5	320000	320000
-3.20E5	-320000	-320000
-3.20E-5	-.000032	0
3.20Elenor	3.2	3
3.20e5	3.2	3
3.20E2.5	320	320

Many MUMPS functions require integers for some of their arguments:

Input To:

function	**argument**	**comment**
$ASCII	1st	identifies the character for which the code is to be returned
$CHAR	all	Like the $SELECT function, $CHAR takes one or more arguments (that is , the maximum number of arguments is not specified). Each argument is the code for a character.
$EXTRACT	2nd, 3rd	starting and ending character positions.
$FIND	3rd	the starting character for the search of a string
$FNUMBER	3rd	number of decimal places for rounding
$JUSTIFY	2nd	width of field within which to right-justify the string in the first argument.
	3rd	number of decimal places for rounding.
$PIECE	3rd, 4th	starting and ending piece positions
$RANDOM	1st	maximum value, plus 1, of random number.

Returned By:

function or operation	**comment**
\	integer division

#	modulo
$JOB special variable	
$STORAGE special variable	
$X special variable	horizontal position of the cursor on the current device.
$Y special variable	vertical position of the cursor on the current device.
$ASCII	the code of the specified character.
$FIND	the character position following the found string, or 0, or -1.
$LENGTH	number of characters or pieces in the first argument.
$RANDOM	a non-negative integer that is less than the argument.

Truth Value

To get a truth value from a string, first determine the numeric value. Once the numeric value is known, then decide if it is zero. If it is zero, it is "false". Anything other than zero is "true". A "false" value is given the character 0 (ASCII 48) and a "true" value is given the character 1 (ASCII 49).
To use MUMPS code to determine the truth value of any string, use the unary operator not (“'”) twice.

Example: `set truth=''string`
Example: 0 or 1

COERCION:

1. determine numeric value
2. if numeric value is zero, truth value is 0
3. if numeric value is non-zero, truth value is 1

EXAMPLES:

string	**numeric value**	**truth value**
12345	12345	1
49.95	49.95	1
-2	-2	1
25Kate	25	1
0.30-	.3	0
+-+-5-	5	1
+18-6	18	1
TEST	0	0
-TEST	0	0
3.20E5	320000	1
-3.20E5	-320000	1
-3.20E-5	-.000032	0
3.20Elenor	3.2	1
3.20e5	3.2	1

3.20E2.5	320	1

INPUT TO:

function	argument	comment
$SELECT	1st	This is actually the first piece (by colon) of every argument of the function.
post conditional		
IF command		

RETURNED BY:

function or operation	comment
=	true if string on left is identical, character for character, to string on right.
[	
]	
>	
<	
&	
!	
'	
$TEST special variable	

Here endeth coercion.

Names

A MUMPS name is a general term that describes the format of several implicit data types. These are described as:

- routine
- label
- variable

A name must begin with either a percent sign (%) or an alpha followed by 0 or more upper-case alpha or numeric. While no explicit limit is placed on the length of a name, only the first eight (8) characters are significant (that is, a standard MUMPS system will ignore any characters beyond the 8th character position).

In this context, the term "variable" means an unsubscripted variable. A variable can be either local or global.

The data types that follow are all address values (sometimes called pointer values). That is, the value gives MUMPS a location, either the location of data (a variable name) or a location of a line of MUMPS code (a routine, label, or a combination). None of these data types are coerced; the complete string is used without modification and must be in the correct format for the context in which it is used.

Label

A label identifies a particular line of code within a routine. A label is not required for every line, but if a label is present on a line it must either be a MUMPS name (see above) or a positive integer.

Format: name or positive integer

EXAMPLES:

A
START
J2
QWE3GY99
236
%DONE
%2D2

INPUT TO:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the line (in the current routine) named by the label. The text will include the label. A space will replace the line-start.
DO command		
GOTO command		

RETURNED BY:

none

Line reference

A line reference identifies a particular line of MUMPS code within a routine by its relative position to a particular label. The format is a label name or a label name followed by a plus sign ("+") followed by a number. The number is the count of lines following the label to reach the line. The number cannot be negative, but may be zero (0).

Format: name or positive integer
or name or positive integer followed by a plus sign followed by a non-negative integer

Examples:

A
START+0
J2
QWE3GY99+88
236+1 Note: This is the 1st line after line 236, not the 237th line.
%DONE
%2D2+2

Input To:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the line (in the current routine) named by the line reference. The text will include the label (if any). A space will replace the line-start.
DO command		
GOTO command		

Returned By:

none

Absolute Line Reference

An absolute line reference can be thought of as a line reference without an explicit label. The format is a plus sign ("+") followed by a number. If each line in a routine was given a unique number starting with the first line (given the number 1) and descending through the routine incrementing by 1, then this number is the number used to identify the line in an absolute line reference.

Format: plus sign followed by a non-negative integer

Examples:

+0
+ 4
+1023

Input To:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the line (in the current routine) named by the line reference. The text will include the label (if any). A space will replace the line-start. If the absolute line reference is "+0," the routine name is returned.

Returned By:

none

Routine Name

A routine name identifies a physical collection of MUMPS code (“physical” to differentiate from “logical”). In all uses of a routine name, the name is preceded with a circumflex (“^”). This lead-in character distinguishes a routine name from a label name.

Format: circumflex followed by a name.

Example:

```
^A
^J2
^DIC
^%DTC
^Z2345678
```

Input To:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the first line of the routine. The text will include the label (if any). A space will replace the line-start.
DO command		
GOTO command		
JOB command		

Returned By:

function or operation	comment
$TEXT1	when the argument of the function is "+0", the name of the "current" routine is returned.

Entry Reference

An entry reference can be either a line reference, a routine, or a combination that specifies a line within a routine. In the latter case, the routine name is concatenated onto the end of the line reference. Note that an entry reference with offset (example label + 2) is not allowed in parameter passing (see Label Reference).

Format:	Line reference
or	routine
or	line reference followed by a routine

EXAMPLES:

A
START+0
^A
J2^DIC
236+l^ %DTC

INPUT TO:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the line named by the line reference within the routine specified. If no routine is specified, the current routine is used. The text will include the label (if any). A space will replace the line-start.
DO command		may not use offset form with parameter passing: see Label Reference.
GOTO command		
JOB command		must include a routine name. may not use offset form with parameter passing- see Label Reference.

RETURNED BY:

none

Label Reference

A label reference is really a subset of the entry reference. The difference is that a label reference may not use a line offset. The reason for the distinction between label reference and entry reference is that parameter passing may only be used with label reference.

Format:	label
or	routine
or	a label followed by a routine

EXAMPLES:

A

START

^A

J2^DIC

236^%DTC

INPUT TO:

function	argument	comment
$TEXT1	1st	causes the function to return the complete text of the line named by the line reference within the routine specified. If no routine is specified, the current routine is used. The text will include the label (if any). A space will replace the line-start.
$TEXT1	1st	causes the function to return the complete text of the line named by the line reference within the routine specified. If no routine is specified, the current routine is used. The text will include the label (if any). A space will replace the line-start.
DO command		required for parameter passing.
GOTO command		
JOB command		must include a routine name, required for parameter passing.

RETURNED BY:

none

Local Variable Name

These are names of local variables without subscripts (although they may name arrays). The variable is in the format described by name. As can seen by the examples, a label, a routine, and a variable may all have the same name. MUMPS knows which one is being referenced by the context of the reference.

Format: name

EXAMPLES:

A
START
J2
QWE3GY99
236
%DONE
%2D2
DIC
Z2345678
Input To:
function
$DATA
$GET
$QUERY

KILL command
LOCK command
NEW command
READ command
WRITE command

Returned By:
none

Local Array Reference

MUMPS multidimensional arrays are identified by a variable name (that is, the name of the array). To reference a cell within the array, a list of subscripts (additional address specifications) are appended to the array name. The subscripts are enclosed in parentheses and separated by commas. A subscript may be any string of printable ASCII characters, 1 to 63 characters long (see SUBSCRIPTS below).

Format: name followed by an open parenthesis followed by a list of subscripts (subscripts are separated by commas; no leading or trailing commas on the list) followed by a close parenthesis.

Examples:

```
A(2)
START("1/1/90","00:00")
J2("QWE3GY99")
QWE3GY99(2,19.2,"0.10")
%DONE("MD","College Park","MUMPS Users'
.....Group")
%2D2(“.”)
DIC(3,0)
Z2345678(“string subscripts!!!!!”)
```

Input To:
function
$data
$GET
$NEXT
$ORDER
$QUERY
KILL command
LOCK command
NEW command
READ command
WRITE command

RETURNED BY:

function or operation	comment
$QUERY	returns either a full array reference or a null

Global Variable Name

These are names of global variables without subscripts (although they may name arrays). The variable is in the format described by name. Global variables can be used interchangeably with the corresponding local version except that globals may not be the argument of either the NEW or READ commands. There may be a label, a routine, a local variable and a global variable all with the same name. MUMPS knows which one is being referenced by the context of the reference.

Format: a circumflex followed by a name

EXAMPLES:

```
^A
^START
^J2
^QWE3GY99
^236
^%DONE
^%2D2
^DIC
^Z2345678
```

INPUT TO:

function or Operator

- $DATA
- $GET
- $QUERY
- KILL command
- LOCK command
- WRITE command

RETURNED BY:

none

Global Array Reference

MUMPS multidimensional arrays are identified by a variable name (that is, the name of the array). To reference a cell within the array, a list of subscripts (additional address specifications) are appended to the array name. The subscripts are enclosed in parentheses and separated by commas. A subscript may be any string of printable ASCII characters, 1 to 63 characters long (see Subscripts). A global array can be used interchangeably with the corresponding local version except that globals may not be the argument of either the NEW or READ commands.

Format: a circumflex followed by a name followed by an open parenthesis followed by a list of subscripts (subscripts are separated by commas; no leading or trailing commas on the list) followed by a close parenthesis.

EXAMPLES:

```
^A(2)
^START(" 1/1/90" ,"00:00")
^J2("QWE3GY99")
^QWE3GY99(2,19.2,"0.10")
^%DONE("MD","College Park”,”MUMPS Users' .....Group")
^%2D2(".")
^DIC(3,0)
^Z2345678("string subscripts!!!!!")
```

Input To:
function or operator
$data
$GET
$NEXT
$ORDER
$QUERY
KILL command
LOCK command
WRITE command

RETURNED BY:

function or operation	comment
$QUERY	returns either a full array reference or a null.

Subscript

Subscripts are often described as strings, but they are actually a separate data type with their own format restrictions. Subscripts may not be greater than 63 characters and may only contain printing ASCII characters (no control characters). A subscript may not be null.

INPUT TO:

none

RETURNED BY:

function or operation	comment
$NEXT	either a subscript or a "-1" (to signal no more subscripts).
$ORDER	either a subscript or a null (to signal no more subscripts).

and The Rest...

The above are the major data types within MUMPS, but they do not make up the complete list. Below is a list of some other unnamed data types requirements that either are expected in a particular context, or are returned by some operation.

INPUT TO:

function or operator	argument	comment
$FNUMBER	2nd	must be an fncode that is a string of characters with each character selected from the following list: "PT,+-"
READ		may use formatting codes (!,#,or ?).
SET command		one or more storage locations followed by an equal sign ("=") followed by an expression.
WRITE command		may use formatting codes(!,#,or ?).
XECUTE command		must be valid MUMPS code.

RETURNED BY:

function or operation	comment
$HOROLOG special variable	two integers separated by a comma.
$DATA	returns a 0, 1, 10, or 11

Expressions

Now that we have looked at how MUMPS handles data typing for each atomic process, we can look at how all this fits together into more complex MUMPS statements. A MUMPS expression is a collection of literals (quoted strings), variables, functions, and operators. Every MUMPS expression produces an explicit data type of string. However, the format of the string will be determined by the operations that produce the string.

Why is this important? Most functions and several commands will take expressions for their arguments. It is up to the programmer to ensure that the expression will produce data in the correct format (data type) for the context in which it will be used.

A simple example:

```
SET RM1="95W"              ; we'll start with two variables,

SET RM2=" 95E"             ; each containing a room number

IF RM1=RM2 DO PROCESS      ; this IF command is checking to determine if the two variables
                           ; are referencing the same room. Here, they are NOT the same
                           ;(The equals sign checks string identity.)
```

```
IF RM1-RM2 DO PROCESS     ; This IF statement does not perform the same check.
                          ; The subtraction operator always takes the numeric value of a
                          ; string, thus the expression becomes 95-95 which is further
                          ; evaluated to O.

IF $E (RM1, $L(+RM1)) ?1A DO PROCESS

                          ;The $LENGTH function will return an integer, which will be
                          ; used to determine the character that is to be returned by the
                          ; $EXTRACT. The $EXTRACT returns a string which is then
                          ; pattern matched to decide if the string is all alphas. The pattern
                          ; match operator returns truth value.
```

Another simple example follows:

```
SET X=7
WRITE X
SET X=7+1
WRITE X
SET X=”7+1”
WRITE X
```

Indirection

Once you understand data typing and expressions, it becomes only a small leap (deep chasm) to indirection. The three kinds of indirection—name, argument, and subscript, each return a different data type. Indirection says "What I'm looking for isn't stored here, but what is stored here is the name of the location where what I'm looking for is really stored", only with less words. This process is called "dereferencing".

Indirection can be done with a variable:

```
    @x
```

or more complex expressions:

```
    @$P(X,U,2,3)
    @("VTX"_($L(NM,Z)-2) )
```

As with the rest of MUMPS, the context of the indirection will determine the data type that is expected to be returned as well as the format of the data that is actually returned.

It is not the intention of this discussion to explain fully the ins and outs, or ups and downs (it is a REALLY deep chasm) of indirection. A discussion of data types in MUMPS will often end up embroiled in expressions and indirection. The point is that a knowledge of MUMPS data types is essential before a programmer can safely take advantage of the power and flexibility that MUMPS provides.

MUMPS is not typeless.

Acknowledgments

The author wishes to thank Gretchen Bradfield and William J. Harvey for their help with this article.

References:

1. Michael Marcotty, Henry F. Ledgard, Programming Language Landscape. Chicago: Science Research Associates, Inc., 1986.
2. Michael Cowlishaw, The REXX Language, Englewood Cliffs, NJ: Prentice Hall.
3. MUMPS Development Committee, X11/88-17 (Proposed 1990 ANSI MUMPS Language Standard)

Endnote

1 This is somewhat misleading since the $TEXT function does not take an expression as an argument but rather is explicitly coded with the line reference. In all other functions an expression (as simple as a literal or as complicated as the programmer wants to make it) can be used in any argument and MUMPS will coerce or assume the correct data type. With the $TEXT function, the form of the argument is not an expression but the actual line reference. The only part of this argument that may be an expression is the "offset" (the numeric portion of the line reference). To use this function in a general way (such as a routine editor that isn't coded to load the same routine each time), the actual function must be built using indirection and a series of concatenations (not recommended for the faint of heart).