M[UMPS] Draft Standard Version 18 (Millennium)

The reader is hereby notified that the following language specification has not yet been approved by the MUMPS Development Committee and that it may be a partial specification which relies on information appearing in many parts of the MDC Standard. This specification is dynamic in nature, and the changes reflected by this approved change may not correspond with the latest specification available.

Because of the evolutionary nature of MUMPS specifications, the reader is further reminded that changes are likely to occur in the specification released herein prior to a complete republication of the MDC Standard.

1. Identification of the Proposed Change

1.1 Title: M[UMPS] Draft Standard Version 18 (Millennium)

1.2 MDC Proposer and Sponsor:

Editor: Ed J.P.M. de Moel
c/o Jacquard Systems Research
800 Nelson Street
Rockville, Maryland 20850-2051
Phone and fax: (301) 762-8999
demoel@jacquardsystems.com

1.3 Motion:

The proposer recommends that this document be accepted by MDC as the current draft standard for X11.1 (as an MDC Type B Document).
Before taking a formal vote on the above motion, it is probably a good idea to take individual votes on the items in the checklist in section 4 below.

1.4 History:

March 2002	X11/TG6/2002-1	Current version, presented for elevation to MDC Type B status, recommendations for votes on non-editorial issues
March 2000	X11/TG6/2000-2	Intermediate version, presented for consideration and discussion on hardhats ftp-site.
February 2000	X11/TG6/2000-1	Intermediate version for review; presented on hardhats ftp-site.
September 1999	X11/1999-7	A list of changes to be applied to the draft standard. Some are editorial and are applied in X11/TG6/2000-1, some will be presented for a vote later.
September 1999	X11/TG6/1999-1	Revised, no votes taken on this document; new editor appointed. Revised: Proposals up through Event Processing; see Editor's Report. Proposed as current draft standard for X11.1.
June 1998	X11/TG6/1998-4	M[UMPS] Draft Standard, Version 14. Superseded X11/TG6/98-1. Revised: adds Editor's Report; folds in editing from Ed de Moel; new extensions: Fix Algoref, Ssvn for User/Group Identification, Sockets Binding, Local Variable Storage, Subscript Indirection & Lock, Miscellaneous Character Functions, and User-Defined Ssvns. Published as reference for current draft standard for X11.1.
March 1998	X11/TG6/98-1	M[UMPS] Draft Standard, Version 13. First complete formalism. Revised: includes all current MDC Type A extensions. Published as reference for current draft standard for X11.1.

1.5 Dependencies

Dependent on all MDC Type A extensions approved at or before the September 1998 meeting in

Seattle.

2. Justification of Proposed Change

2.1 Needs

The MDC needs an up-to-date draft standard both to be ready for submission of the next standard to ANSI and to give proposers an up-to-date baseline for suggesting changes.

2.2 Existing Practice in Area of the Proposed Change

Does not apply for this document.

3. Description of Proposed Change

3.1 General Description of the Proposed Change

This document contains X11.1!1995 as modified by all current MDC Type A extensions approved before the cut-off decision (September 1998).

3.2 Annotated Examples of Use

Not relevant.

3.3 Formalization

See below.

4. Checklist

Seq	Description	Yes	No	Abs
1	The modifications applied to X11/SC13/97-9: Mathematics Errors are indeed of an editorial nature			
2	The modifications applied to X11/SC13/TG6/98-3: Horolog System Function are indeed of an editorial nature			
3	The modifications applied to X11/SC15/98-5: Error Handling Corrections are indeed of an editorial nature			
4	The modifications applied to X11/SC15/TG2/98-2: Object Usage are indeed of an editorial nature			
5	The modifications applied to X11/SC12/98!13: User-Definable I/O Handling are indeed of an editorial nature			
6	The modifications applied to X11/SC15/98-8: $MUMPS Function are indeed of an editorial nature			
8	The modifications applied to X11/98-24: Duplicate Keywords Clarified are indeed of an editorial nature			
11	The modifications applied to X11/98-27: Pattern Match String Extraction are indeed of an editorial nature			
12	The modifications applied to X11/98-28: Event Processing are indeed of an editorial nature			
13	The modifications described in X11/1999-7: Notes about the draft standard, first section are indeed of an editorial nature			
13/4	The definition of **character** (in section 4) is currently as intended.			

Seq	Description	Yes	No	Abs
13/10	The definition of **diacritic** (in section 4) is currently as intended.			
13/16	The definition of **letter** (in section 4) is currently as intended.			
13/252	Should the unused columns be removed from table A-1 (ASCII, single level collation)?			
13/17	It was appropriate to insert the word **usually** in the definition of **level** (in section 4).			
13/19	Replace the definition of modulo as indicated in X11/1999-7.			
13/32	Should the definition of the RSAVE command be modified to specify that errors M21 and M57 could result from execution of that command? M21 = name occurs more than once in formallist M59 = duplicate label			
13/35	Modify text in section 6.3 as indicated in X11/1999-7.			
13/84	Insert a specification of a model for the LOCK table?			
13/159	Change text in 8.1.6.1 as indicated in X11/1999-7.			
13/194	Insert text about CLOSEing any devices in the specification of the HALT command?			
13/340	Use M[UMPS] instead of M?			
14	The changes itemized in this section are indeed editorial.			
15	The modifications applied to X11/1998!19: User-Defined structured system variables are indeed of an editorial nature			
16	The modifications applied to X11/1998-29 Local Variables in ^$JOB are indeed of an editorial nature			
17	The modifications applied to X11/SC13/1998-10 $%FORMAT^STRING are indeed of an editorial nature			
18	The modifications applied to X11/SC13/TG3/1999-4 Data Record Functions are indeed of an editorial nature			
19	The modifications applied to X11/SC13/TG6/98!3: $HOROLOG Function are indeed of an editorial nature			
20	The modifications applied to X11/SC15/1998-113: Generic Indirection are indeed of an editorial nature			
20a	Add specification of error code to definition of Generic Indirection?			
E001	Apply the modification suggested in the “Editor’s Note” on page 10			
E002	Delete the phrase identified in the “Editor’s Note” on page 36			
E003a	Add the phrase suggested in the “Editor’s Note” on page 70 with ecode = M28			
E003b	Add the phrase suggested in the “Editor’s Note” on page 70 with ecode = M9			

Seq	Description	Yes	No	Abs
E004	Replace “imparted” by “created” in dspecification of UNBLOCK command on page 96, 100?			
E005	It is correct that the colon in the definition of recordfieldglvn on page xxvi, 115, 116 is within the brackets.			
E006	The characters with codes 178=², 179=³, 185=¹ 188=¼, 189=½ and 190=¾ should also match pattern code N (numeric), see Editor’s Note on page 154, 155, 156.			
E007	Apply the modifications to $%LOWER^CHARACTER that are suggested in the Editor’s Note on page 177			
E008	Apply the modifications to $%PATCODE^CHARACTER that are suggested in the Editor’s Note on page 178			
E009	Apply the modifications to $%UPPER^CHARACTER that are suggested in the Editor’s Note on page 178			
E010	Addition of ^$SYSTEM(system,”FORMAT”) and ^$JOB(job,”FORMAT”) is appropriate (see pages 34 and 35)			
E011	Formalism of “generic indirection” brought in alignment with examples. Examples were indeed intended behavior (See page xxxvi)			
E012	New text for M95 is appropriate (see page 156)			
E013	New text for M99 is appropriate (see page 156)			
E014	Suggestion for trigger for error M21 is appropriate (see page 11)			
E015	New text for M21 is appropriate (see page 155)			
E016a	Error for hyperbolic cotangent of 0 should be M9 (see page 69)			
E016B	Error for hyperbolic cotangent of 0 should be M28 (see page 69)			
E017	Definitions from X11.6 that are referenced in X11.1 should appear in footnotes (see page 100)			
E018	Insertion of note about error M57 in 8.1.6.1 is appropriate (see page 11)			
E019	Destination of Set $Qsubscript should be glvn V namevalue (see page 115)			
E020	$%LOWER should be part of library CHARACTER (not STRING) (see page 65)			
E021	$%UPPER should be part of library CHARACTER (not STRING) (see page 66)			
E022	Example for use of algoref (collation) is appropriate (see page 27)			
E023	Example for use of algorefs (LOWER and UPPER) is appropriate (see page 28)			
E024	Suggested computational equivalent for $REVERSE is appropriate (see page 59)			
E025	$%PATCODE should be part of library CHARACTER (not STRING) (see page 65)			

Seq	Description	Yes	No	Abs
E026	Suggestion to change the term “dual” in specification of binary relational operators to “multi-character” is appropriate (see page 79)			

Table of Contents

Notes from the Document Editor

This document is ANSI X11.1!1995 *plus* the following MDC Type A extensions:

Number	Final #	Note	Name	MDC A	Depends On
M Draft Standard Version 11					
93-39	93-39		$REFERENCE	93 Jun	
94-4	94-4		Two-Character Operators	93 Jun	
94-5	94-5		Initialising Intrinsics	93 Jun	
94!14	94!14		Multiple patatoms Within alternation	94 Feb	
94-23	94-23		Library Proposal	94 Jun	
90-29	90-29		Complex Numbers	90 Sep	
94-28	94-28		Portable String Length	94 Jun	
94-46	94-46		^$GLOBAL Correction	94 Jun	
94-47	94-47		New svn Addition: $TEST	94 Jun	
95-2	95-2		Execution Environment	95 Jan	
95!11	95!11		Library Functions - General Math	95 Jan	94-23
95!12	95!12		Library Functions - Trigonometry	95 Jan	94-23
95!13	95!13		Library Functions - Hyperbolic Trig	95 Jan	94-23
95!14	95!14		Library Functions - Complex Math	95 Jan	94-23
95!18	95!18		Number of Subscripts in gvn	95 Jan	
95!19	95!19		Leading Zero Interpretation in $FNUMBER	95 Jan	
95-20	95-20		Sign of Zero Interpretation in $FNUMBER	95 Jan	
95-21	95-21		SET Command Clarification	95 Jan	
95-22	95-22		"Standard" in Library Element Defs	95 Jan	94-23
95-31	95-31		Kill Indirection	95 Jan	
95-63	95-63		Naming String Length Error	94 Feb	
SC12/93-33			Effect of CLOSE $IO	93 Jun	
SC13/93-36	96-34		Modulo by Zero	93 Jun	
SC13/94-33			Kill Data and Kill Subscripts of glvns	94 Jun	
95-96	95-96		Spaces at the End of a line	94 Feb	
SC15/95-5	96-65		Normalize Definition of TSTART	95 Jan	
95-94	95-94		Parameter Passing Clarification	93 Oct	
95-95	95-95		Portable controlmnemonics/ mnemonicspaces	95 Jun	
95-91	95-91		$ORDER Definition	95 Jun	
95!116	95!116		^$JOB Device Information	95 Jun	
95!117	95!117		ssvn Collation	95 Jun	
95!118	95!118		Undefined ssvns	95 Jun	
95!119	95!119		Extended extids	95 Jun	
95!111	95!111		PRODUCE Library Function	95 Jun	94-23
95!112	95!112		REPLACE Library Function	95 Jun	94-23
M Draft Standard Version 12					
95!132	95!132		Parameter Passing to a Routine	95 Oct	
95!136	95!136		String Length Limit Exceeded	95 Oct	
95!137	95!137		“Backward Compatible” & “Reserved”	95 Oct	
96-7	96-7		Lower-case Characters in Names	95 Oct	
96-9	96-9		Pattern Negation	95 Oct	
96!10	96!10		Reverse $QUERY	95 Oct	
96!11	96!11		fncode Correction	95 Oct	
96!13	96!13		Portable Length Limit of names	95 Oct	96-7
95-88	96-45		charset Names	95 Oct	
96-51	96-51		Device Environment	95 Oct	

Number	Final #	Note	Name	MDC A	Depends On
96-52	96-52		Routine Management	95 Oct	
SC12/96-5	96-44		Improve mnemonicspace Handling	96 Mar	
SC13/96-2	96-41		*String* and *M* Collation	96 Mar	
SC13/96-3	96-42		charset: ISO-8859!1-USA	96 Mar	95-88 & SC12/96-2
SC13/95-27	96-32		Sign of Zero in $FNUMBER	96 Mar	
SC13/TG5/96!1	96-26		Library Functions - Matrix Math	96 Mar	94-23
SC13/TG5/96-2	96-27		XOR Operator	96 Mar	
SC15/96!1	96-49		QUIT with Argument in FOR	96 Mar	
SC15/96-4	96-43		ssvn Formalization	96 Mar	94-23
SC15/96-8	96-57		GOTO Rewording	96 Mar	
SC15/96-9	96-58		Add JOB to Routine Execution	96 Mar	
SC15/96-5	96-35		Parameter Passing Cleanup	96 Mar	
M Draft Standard Version 13					
96-74			Operator Overrides	97 Feb	
96-67			Leading Zero in $FNUMBER	97 Apr	
96-68			Negative Subscripts in nameval	97 Apr	
97-3			Pattern Ranges	97 Nov	
97!10			mnemonicspec Cleanup	97 Feb	
97-22			SET $QSUBSCRIPT	97 Aug	
97-23			Portable Length Limit of Strings	97 Aug	
97-25			First Line Format	97 Nov	
97-31			Output Timeout	97 Nov	
M Draft Standard Version 14					
98-5			Fix Algoref	98 Mar	
98-8			Ssvn for User/Group Identification	98 Mar	
98!14			Sockets Binding	98 Mar	
SC12/98-4	98-21		Miscellaneous Character Functions	98 Mar	
SC13/97-8	98!19		User-Defined Ssvns	98 Mar	
SC13/TG15/97-3			Local Variable Storage	98 Mar	
SC15/96!13			Subscript Indirection & Lock	98 Mar	
M Draft Standard Version 15					
SC13/97-9	**T**	1	Mathematics Errors		
SC13/TG6/98-3	**T**	2, 19	Horolog System Function		May
SC15/98-5	**T**	3	Error Handling Corrections		May
SC15/TG2/98-2	**T**	4	Object Usage		May
SC12/98!13	**T**	5	User-Definable I/O Handling		July
98-23	**T**	7	OPEN Command Clarification		Aug
98-24	**T**	6	Duplicate Keywords Clarified		Aug
98-25	**T**	7	Device Parameter Issues		Aug
98-26	**T**	8	Canonic Form of ssvn Name		Aug
98-27	**T**	9	Pattern Match String Extraction		Aug
SC15/98-8	**T**	10	$MUMPS Function		July
98-28	**T**	11	Event Processing		Aug
98-29	next	12	Local Variables in ^$JOB		Aug
98-30		?	NEW $REFERENCE		Aug
SC15/1998-7		?	IF THEN ELSE		Aug
SC12/98!11		?	Output Timeout Initialized		Aug
SC12/98!14		?	Undefined Devicekeyword		Aug
SC13/98!13		?	Define Variable M		Aug
SC13/98!15		?	Definition Reverse $QUERY		Aug
M Draft Standard Version 16					

Number	Final #	Note	Name	MDC A	Depends On
X11/1999-7		13	Notes by Ed de Moel Editorial corrections applied		
M Draft Standard Version 17					
		14	Changes to the format of the document		
SC13/98!10		17	Format Library Function		July
SC15/98!11		20	Generic Indirection		Aug
98-32		?	Cyclic Redundancy Code Function		Oct
98-29		16	Local variables in ^$JOB		
SC13/TG3/98-4		18	Data Record Functions		
M Draft Standard Version 18					
		21	Several typographical errors		
		22	Rewording of error codes 95 and 99		
		23	Suggestions for error codes 21. 39, 46, 47 and 113		
		24	Attempt to deal with definitions that are not part of X11.1, but appear in X11.6		
98!14		25	Insert remainder of specification of sockets binding		
		26	Additional suggestions from the editor		

Dear MDC Voting Member,

This version of the M draft standard (MDS) does not have the final rounds of polishing with regard to 1) the index, 2) the overall formatting, 3) language consistency, and 4) my comments on what changes I had to make to the extensions in the editing process. I have included complete notes on those extensions added with this version of the MDS. The rest of the comments and polishing will be included with the next version of the MDS.

If you have any questions or suggested corrections please contact me.

Sincerely,

Ed de Moel
demoel@jacquardsystems.com

1. X11/SC13/1997-9: Mathematics Errors

This extension had one crucial point of confusion (My thanks to Art Smith, who helped me figure out the proper interpretation of the confusing sentence), and one oversight, plus some minor naming decisions:

> 1. The text describing overflow errors was extremely confusing (as judged by the Editor and the MDC Chair). We came up with an interpretation of the sentence that accounted for all of its terminology, then reworded it to make that interpretation as clear as possible.
>
> Before the change, the text for overflow errors read:
>
> > If the result of any mathematical operation is too large (either positive or negative) for the implementation to represent to the accuracy specified earlier in this clause, an error will occur, with ecode equal to M92.

After the change, it reads:

> If the result of any arithmetic operation is too large (either positive or negative), or if it is too large for the implementation to represent with the accuracy specified in the previous paragraph, an error condition occurs with ecode = "M92".

2. No text for the error codes was given for inclusion in the error codes annex. In the absence of any guidance, the following were selected (& added to the index):

M92	arithmetic overflow
M94	0 raised to the power of 0
M95	complex number

3. One minor ambiguity resulted from adding text defining ecode = "M95" to handle complex results but leaving in place the text saying results producing complex numbers are not defined. I decided to strike the old language as being superseded by the new.

Before:

** produces the exponentiated value of the left operand, raised to the power of the right operand. Results producing complex numbers (eg, even numbered roots of negative numbers) are not defined.

After:

** produces the exponentiated value of the left operand, raised to the power of the right operand. On an attempt to compute 0 * * (a negative number), an error condition occurs with ecode = "M9". On an attempt to compute 0 * * 0, an error condition occurs with ecode = "M94". On an attempt to compute the result of an exponentiation, the true value of which is a complex number with a non-zero imaginary part, an error condition occurs with ecode = "M95".

2. X11/SC13/TG6/1998-3: Horolog System Function

Only minor changes:

1. Replaced "This gives" with "This form gives" to make it match the other functions.

2. Replaced "intexpr" with "intexpr".

3. Moved definition of variables up before "The following cases...", introduced language like "Let *D* be an integer...", and italicized the variables, all to match similar definitions elsewhere, especially the $HOROLOG intrinsic special variable.

4. Adjusted spaces, commas, capitalization, et cetera, corrected the lettering of the cases, and introduced "If " to each case for consistency.

5. Replaced "$H" with "$HOROLOG" for consistency.

6. Table of Contents changes:
   Added $HOROLOG function as clause 7.1.5.8 & renumbered.

7. New index entries:
   $HOROLOG function currently mingles with entries for the intrinsic special variable. We'll split them in sub-entries during the index cleanup.

3. X11/SC15/1998-5: Error Handling Corrections

A little rewording, reorganizing, and interpretation, but basically sound:

1. In clause N2.1.7 and N2.1.8, I standardized the wording describing error M101, and added it to the error codes annex:

Before: An "M101" error is generated instead.

After: Instead, an error condition occurs with ecode = "M101".

And in the annex & index: M101 invalid $ECODE value

2. In clause N2.1.9 and N2.1.10, text discussing maximum string length is inserted into Section 1, which nowhere else discusses such limits. All such discussion is in Section 2 so I moved this text there as well. I inserted the text as a new paragraph in 2.9 Special Variables, and modified it to compensate for the relocation.

Before (in 6.3.2 Error Processing of Section 1)

If appending to $ECODE or $STACK($STACK,"ECODE") would exceed an implementation's maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,"ECODE") to discard.

After (in 2.9 Special Variables of Section 2):

If appending the information about a new error condition (See 6.3.2 of Section 1) to $ECODE or $STACK($STACK,"ECODE") would exceed an implementation's maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,"ECODE") to discard.

3. For consistency, I also replaced the phrase "error event" with "error condition" in the first paragraph of the last part of this change.

4. I reworded the definition of the new paragraph defining an Error Processing Transfer of Control because as it stood the new sentence defined the whole thing to be only the first part of the transfer, the termination of active states. I also replaced "text of the first list" with "body of the first line" and introduced "body of the" before "the second line".

Before:

"An Error Processing transfer of control consists of terminating the current command and processing in the scope of any active FOR commands and indirection. Execution explicitly resumes at the same LEVEL with two lines where the text of the first list is the value of $ETRAP and the second line is:"

After:

"An Error Processing transfer of control consists of first terminating the current command and processing in the scope of any active FOR commands and indirection; and second, explicitly resuming execution at the same LEVEL with two lines where the body of the first line is the value of $ETRAP and the body of the second line is:"

5. Another problem with this change is that the results of the appendix included to show the changes in context does not agree with the formalism. Specifically, the formalism keeps the full metalanguage for the two inserted lines, carefully modifying them to match the description in the previous paragraph and appending an explanation of li. The Appendix, however shows these lines deleted, and ends up with an invalid statement (that the second line is QUIT:$QUIT "" QUIT,

something that can at most be a linebody, not a full line). Given the choice, I'm sticking with the formalism and interpreting the annex as in error. Further, I'm inserting a phrase to make this interpretation explicit: "The two lines are:"

4. X11/SC15/TG2/1998-2: Object Usage

A few oversights and rewordings, but most of the work was simple but laborious indexing, glossary-building, and similar overhead, not unexpected for such a radical extension:

1. No Glossary clauses were created by this standard, but due to the extensive terminology introduced by this proposal, many such clauses are needed. Further, this extension makes some existing glossary entries inaccurate, such as object and type. I created the following using the formalism and general description of the extension for guidance (& added them to the index or cleaned up their index entries):

4.A. call by name: A calling program names an actual parameter and passes its value to an object's service. Limited to a single value, that is, the value of a scalar variable or of one node in an array. See also call by reference, call by value.

4.B call by reference: A calling program passes a reference to its actual parameter. If the called subroutine, function, or object's service changes its formal parameter, the change affects the actual parameter as well. Limited to unsubscripted names of local variables, either scalar or array. See also call by name, call by value.

4.C call by value: A calling program passes the value of its actual parameter to a subroutine, function, or object's service. Limited to a single value, that is, the value of a scalar variable or of one node in an array. See also call by name, call by reference.

4.D. default property: the default state of an object; a property to which an OREF evaluates if used in a property reference that doesn't name a specific property. See default state, OREF, property.

4.E: method: a service that represents the behavior that may be requested of an object. See object, property, service.

4.F MVAL: The type of any data value that may be represented as a string of variable length. Arithmetic operations interpret strings as numbers, and logical operations further interpret the numbers as true or false. See also truthvalue, type.

4.G object: An identifiable, encapsulated software entity whose state and behavior can only be observed or changed by use of its services. An object is considered as a whole in relation to other entities, and is identified by a value of data type OREF. See OREF, service.

4.H OREF: An object reference. A value of data type OREF is a reference to an object that uniquely identifies that object. OREFs have no literal representation. Under most circumstances, values of data type OREF are coerced into values of type MVAL based on the value of the default property of the object identified by the value of data type OREF. See default property, object, type.

4.I parameter (of a function, subroutine, or object's service): The calling program provides actual parameters. In a called function or subroutine formal parameters relate by position to the caller's actual arguments. In a called object's service formal parameters can relate by position or name to the caller's actual arguments. See also call by name, call by reference, call by value, parameter passing.

4.J parameter passing: This alliterative phrase refers to the association of actual parameters with formal parameters when calling a subroutine, function, or object's

service.

4.K. property: a service that represents the external view of some of an object's data. An object may have a default property. See default property, method, object, service.

4.L. service: bodies of code associated with objects. Services are the only mechanism through which the state of an object may be altered. An object's services include methods and properties. See method, object, property.

4.M type: M recognizes only two data types, the object reference, or OREF, and the string of variable length, or MVAL. See MVAL, OREF.

2. One decision required was where to place the Object Usage clause. The Values clause is clearly and correctly placed before clause 7.1.1. Variables. Nevertheless, although the Object Usage clause occurs first in the extension document and defines metalanguage elements the Values clause uses (oref and mval), it must be placed later.

Part 6. Routine is incorrect because it emphasizes the internal structure of routines, whereas Object Usage says nothing about the internal structure of objects or their services. It describes instead how to call those services. Since the closest existing topics are found in clauses 8.1.6. Line reference and 8.1.7. Parameter passing, I placed Object Usage immediately after 8.1.7 as clause 8.1.8. Object usage.

Since X11.1 tends to put more self-sufficient metalanguage definitions earlier and interdependent ones later, I'm moving the definitions of the oref and mval metalanguage elements back to the Values clauses which use them and where they are more fully explained.

3. The change adding ASSIGN to the list in clause 6.3.2. Error processing of the ways $ECODE can change has two problems. First, David Marcus's Error Handling Corrections extension (X11/SC15/98-5) modifies the language here to add another reference to this list in the prior paragraph, so at the minimum this change is affected by that.

However, more fundamentally, I believe this change is a leftover from when this proposal redundantly included the characteristics of the SET command in the ASSIGN command. More recent versions of this proposal removed the redundancy so that now the ASSIGN command can only set OREF values.

$ECODE does not accept OREFs as values. The Error Handling Corrections extension now defines as error "M101" trying to assign a value to $ECODE that does not conform to the required format. This means the change to add ASSIGN to this list would produce a false statement, since trying to ASSIGN anything to $ECODE would generate an error condition with $ECODE = "M101" instead of the value of expr as defined for an Error Processing transfer of control.

Therefore, I'm interpreting this change as being an invalid leftover from an earlier draft and as conflicting with Error Handling Corrections, so I am not applying it to the draft standard.

4. When clause 8.1.8.1 discusses the use of names that do not conform to the syntax of a name, it explicitly introduces this as applying to both service names and named actual argument keywords, but in the next sentence refers only to services. I corrected this for consistency:

Before: In these cases, the name of the service must be represented as a strlit,

After: In these cases, the name of the service or parameter must be represented as a strlit,

5. Missed an essential change in the portability section:

Before:

The M Language Specification defines a single data type, namely, variable length character

strings. Contexts which demand a numeric, integer, or truth value interpretation are satisfied by unambiguous rules for mapping a string datum into a number, integer, or truth value.

After:

The M Language Specification defines two data types, namely, MVALs (variable length character strings) and OREFs (object references). Contexts which demand a numeric, integer, or truth value interpretation are satisfied by unambiguous rules for mapping an MVAL into a number, integer, or truth value.

6. add M105, M106, M107, M108 to annex & index:

M105	Inaccessible Object
M106	Invalid Service
M107	No Default Property
M108	Not an Object

7. add new metalanguage to annex & index:

actualkeyword	actual argument keyword
assignargument	ASSIGN argument
assigndestination	ASSIGN destination
assignleft	ASSIGN left
fservice	service name with parameters
mval	M value (string)
namedactual	named actual argument
namedactuallist	named actual argument list
object	expression atom, value interpreted as an OREF
oref	object reference value
owmethod	object with method
owproperty	object with property
owservice	object with service
servicename	service name

8. Table of Contents:

Rather than renumber all of section 7, Values is folded in with Variables, so:

7.1.1 is now Values and Variables
7.1.1.1. is Values
7.1.1.2. is Variables

$TYPE ends up in clause 7.1.5.22 & the rest move down.
Object Usage ends up as 8.1.8 & User-Defined mnemonicspaces moves to 8.1.9.
ASSIGN ends up in clause 8.2.1 & the commands are renumbered.

5. X11/SC12/1998!13: User-Definable I/O Handling

One episode of rewording and interpreting plus the usual editorial overhead:

1. Add new metalanguage to annex & index:

ffformat	form feed format code
iocommand	I/O command
nlformat	new line format code
positionformat	position format code
tabformat	tab format code

2. This extension left clause 8.1.9 (User-defined mnemonicspace) in a self-contradictory state. The following two paragraphs, the first introduced by this extension and the second by an earlier extension, directly clash:

> Upon completion of execution of a routine associated with a user-defined mnemonicspace, the naked indicator and $TEST are restored to their original values.

> Note: It is the responsibility of the user-defined mnemonicspace routine to process the deviceparameters in the appropriate order and return $TEST appropriately in the event that a timeout is present.

As is clear from the document editor's comments in the laudably complete section 7 of the proposal, the Task Group wrestled with the problem of what to do with $TEST, reversing itself at least twice at the end.

When the comments from October, 1995 are factored in, it seems the extension should have preserved $TEST except when a timeout is included, when $TEST is left unpreserved so the programmer of the mnemonicspace routine can set $TEST based on whether the action times out. This seems to me the least intrusive way of reconciling the two paragraphs as well, though even with that interpretation the actual existing text is still too contradictory.

Further, the language of the second paragraph can be read as implying that it is only the responsibility of the user-defined mnemonicspace routine to process the deviceparameters in the appropriate order *in the event that a timeout is present*. Therefore, to solve the contradiction and the ambiguity I have reworked the paragraphs as follows:

> It is the responsibility of the user-defined mnemonicpace routine to process the deviceparameters in the appropriate order.
>
> Upon completion of execution of the user-defined mnemonicspace routine, the naked indicator is restored to its original value.
>
> In the event that a timeout is *not* present, $TEST is also restored when execution of the routine completes. However, if a timeout is present $TEST is *not* restored and it is the responsibility of the user-defined mnemonicspace routine to return $TEST to indicate whether the operation times out.

I then reorganized the paragraphs at the end of this clause to group them by whether they discuss the parameters passed in, what happens during execution, and what happens after execution.

6. X11/SC15/1998-8: $MUMPS Function

Add $MUMPS to Table of Contents & renumber the rest of the functions. Add mumpsreturn and noncommasemi to Metalanguage Dictionary. Add ecode S0 to the Error Code Annex. Index.

7. X11/1998-23: OPEN Command Clarification

Perfect.

8. X11/1998-24: Duplicate Keywords Clarified

Almost perfect. I put in the metalanguage unchanged, but the instructions for RSAVE are incorrect. RSAVE shares definitions with RLOAD and does not repeat them, so for example there is no paragraph starting "All values of routinekeyword..." in the RSAVE clause; that's in the RLOAD clause. Ideally, RSAVE would probably have had added a paragraph like those for OPEN and USE, in this case pointing to the definitions in RLOAD rather than repeating them. However, the added text in RSAVE is not inaccurate and is still clear, if slightly redundant, so I opted for the least intrusive approach and added the paragraph unchanged just after the metalanguage in RSAVE.

9. X11/1998-25: Device Parameter Issues

Perfect.

10. X11/1998-26: Canonic Form of ssvn Name

Perfect.

11. X11/1998-27: Pattern Match String Extraction

Perfect. Added patsetdest to the Metalanguage dictionary. Indexed.

12. X11/1998-28: Event Processing

Almost perfect. I just needed to come up with the glossary items, the text for the error codes, and the text for the metalanguage element dictionary, and insert three subheaders in the new clause 6.3.3 to help readability. Mainly just the usual editorial tasks.

1. added new Glossary entries. I discovered that the Glossary is also deficient in entries defining terminology from the other three processing modes, so I added those here too:

> 4.aa event processing: one of three special processing modes within a job. In response to a registered, enabled event, the job transfers control to that event's handler if the job has started event processing. See asynchronous, call-back processing, event, event class, event handler, synchronous, routine execution. See also 6.3.3 Event processing.
>
> 4.ab event: a registered occurrence outside the normal program flow. Events are uniquely identified and tied to an event handler. See event handler, event processing.
>
> 4.ac event handler: a sequence of commands identified by a label reference and registered as the code to execute in response to a specific event. See event processing.
>
> 4.ad call-back processing: the execution within an event handler in response to synchronous processing of an event. Call-back processing follows the rules of normal routine execution, but when the event handler terminates, the job returns to synchronous processing of events. See event processing.
>
> 4.ae event class: a set of events for which event processing must be started or stopped together and in the same mode (synchronous or asynchronous). See event processing.
>
> 4.af synchronous event processing: the processing of events such that the job's execution suspends while it waits for events. See event processing.
>
> 4.ag asynchronous event processing: the processing of events such that the job continues normal routine execution while it waits for events. See event processing.
>
> 4.ba routine execution: the normal program flow within a job, in which routines are executed in blocks of code. A job can also execute M code in three special processing modes. See block, error processing, event processing, job, routine, transaction processing. See also 6.3 Routine execution.
>
> 4.bb routine: the primary storage unit for M code, structured as a named sequence of lines of code. See routine execution.
>
> 4.ca error processing: one of three special processing modes within a job. In response to an error, a job transfers execution to an error handler and/or terminates the current block of code. See error, error handler, and routine execution. See also 6.3.2 Error processing.
>
> 4.cb error: an abnormal condition arising within the current job and identified by an error code. When an error occurs, the job's execution mode changes to error processing. See error processing.

4.cb error handler: a sequence of commands executed when an error occurs. See error processing.

4.da transaction processing: one of three special processing modes within a job. From the time a transaction starts until it is committed, other processes cannot access the global variable modifications made within the transaction. See routine execution, transaction. See also 6.3.1 Transaction processing.

4.db transaction: the execution of a sequence of commands that begins with an explicit start and ends with either a commit or a rollback. A transaction is atomic, consistent, isolated, and durable. When a transaction start is executed, the job's execution mode changes to transaction processing. See transaction processing.

2. added new error codes:
m110 insufficient resources

Editor's note:
Cannot find this added error code, nor any reference to it. Could possibly M98 (Resource unavailable) be intended?

m102 events cannot be both synchronous and asynchronous
m103 invalid event
m104 invalid event id

3. added new metalanguage elements to the dictionary:

einfoattribute	event information attribute
evclass	event class
eventexpr	event expression
evid	event id

4. added new subsections under 6.3.3:
6.3.3.1 Event classes
6.3.3.2 Asynchronous event processing
6.3.3.3 Synchronous event processing
and shifted last paragraph in 6.3.3 up before 6.3.3.1.

5. moved the addition to the definition of ssvn to the definition of ssvname (this section was altered by a previous extension) and reworded it:
from: "syntax of ^$EVENT structured system variable"
to: "E [VENT]"

6. I added another subclause to the new clause on event processing in Section 2: Portability. The new subclause lists the event classes that may not be portable:

12.4 Event classes
Use of the following event classes may not be portable: COMM, INTERRUPT, POWER, and Z[unspecified]. Use of HALT event classes where evid does not equal 1 may not be portable.

The editor is somewhat uncomfortable with the proliferation of portability commentary outside the Portability section, but ultimately decided it would be clearest to readers of X11.1 for us to leave it as specified in the extension. The editor also debated strengthening the language to "is not portable," but noticed this is the wording used in VIEW and $VIEW and that BREAK has no discussion of portability, and so opted to leave the question of standardizing how we express this for another day.

7. Added ^$EVENT as the new section 7.1.3.3, and renumbered the rest of the ssvn clauses.

8. The document author and I debated how best to word the two instances where the syntax and

semantics of metalanguage elements added to X11.1 by this extension are defined in X11.6. We agreed that none of that information is needed to use the M event processing features without MWAPI, and that the purpose of including those metalanguage elements in this extension is to keep the two documents compatible with one another. The only change I felt necessary was to emphasize this fact. I shifted the order of the values of eventexpr to put EVENTDEF first, and added text to the existing note to clarify that these nodes are used for MWAPI information:

> Nodes under ^$EVENT(einfoattribute) are used to identify specific behavior of MWAPI events.

I made a similar change to the definition of ETRIGGER, putting ^$JOB before ^$WINDOW and adding this text to the note about espec and einforef:

> ETRIGGER arguments that evaluate to nodes under ^$WINDOW are used to cause MWAPI events to occur.

9. I reformatted the text for ^$EVENT to match that of the other ssvns. I inserted subheaders introducing each node under ^$EVENT and rearranged the text to fit. This required that I choose how to express the values of these nodes, which I did as either intexpr or tvexpr based on the descriptive text. I replaced some constant string literal references to subscript values with expr V value constructs. I also fixed a sentence splice at the end of the explanation of the INTERVAL node.

10. I did a similar reformatting job on the new text for ^$JOB (now in 7.1.3.5.5). I changed the value of the first of these new nodes, which is settable, from entryref to expr V labelref.

11. I added the new commands to the definitions of commandword, and added ABLOCK to the definition of command.

12. I added clauses with the following numbers for the new commands, and renumbered the rest:
- 8.2.1 ABLOCK
- 8.2.3 ASTART
- 8.2.4 ASTOP
- 8.2.5 AUNBLOCK
- 8.2.10 ESTART
- 8.2.11 ESTOP
- 8.2.12 ETRIGGER

13. While renumbering the commands to make room for the 7 new ones, I standardized the formatting, punctuation, and indexing of the existing ones. Step d of the execution of the QUIT command I reorganized a little more carefully than the rest because of an ambiguity I noticed that is essentially a formatting problem. Thanks go to the MDC chair for helping me feel out the meaning of this clause, where the formatting problem was, and how best to express it.

Before:

> d) If the frame contains formal list information, extract the formallist and process each name in the list with the following steps:
>
> > 1) Search the NAME-TABLE for an entry containing the name. If no such entry is found, processing of this name is complete. Otherwise, proceed to step 2.
> >
> > 2) Delete the NAME-TABLE entry for this name.
>
> Finally, copy all NAME-TABLE entries from this frame into the NAME-TABLE.
>
> Processing of this frame is complete, continue at step b.

After:

> d) If the frame contains formal list information:
>
> > 1) Extract the formallist and process each name in the list with the following steps:

i) Search the NAME-TABLE for an entry containing the name. If no such entry is found, processing of this name is complete. Otherwise, proceed to step ii.

ii) Delete the NAME-TABLE entry for this name.

2) Finally, copy all NAME-TABLE entries from this frame into the NAME-TABLE.

3) Processing of this frame is complete, continue at step b.

14. I also revised the metalanguage of the event processing commands to match that of NEW and KILL, which they resemble in their three forms.

13. X11/1999-7: Notes about the draft standard

This document contains a number of recommendations to apply changes to the draft standard. Changes that are of an editorial nature are applied in this iteration of the document. Changes of a more substantive nature will be presented and will be voted on individually.

The changes that are deemed of an editorial nature are (numbers apply to X11/1999-7):

1. Use official name of document
2. Make text explicit
3. Make text explicit
5.Remove unclear reference
6. Make text more explicit
7. Typo
8. Consistent word usage
9. Consistent word usage
11. Missing quote character, capitalization
12. Make text more explicit
13. Replace all occurrences where “global” is used as a substantive by “global variable”. (Common MUMPS usage is well known, but the standard should use more formal language.)
14. Consistent word usage
15. Be consistent with internationalization
16. Be consistent wirh internationalization
18. Be consistent with internationalization
20. Change mandated by Interpretation Taskgroup
21. Missing one of the possible meanings
22. Typo, missing word
23. Change mandated by Interpretation Taskgroup
24. Formatting issue
25. Cut and paste issue: different detail in this instance
26. Part of including Type A “Object Usage”, already included in Draft # 15
27. Already included in Darft # 15
28. Formatting issue
29. Make text more explicit
30. Formatting issue and ramifications of internationalization
31. Make specification more explicit
33. Indefinite article is better
34. Make text more explicit
37. Make text more explicit
38. Formatting issue, already included in Draft # 15
39. Partly, already included in draft # 15, makes text more explicit
40. Consistent word usage
41. Formatting issue
42. Consistent word usage
43. See 42, consistent singular/pulural usage
44. Formatting issue

45. Consistent usage
46. Consistent word and metalanguage element usage
47. Formatting issue
48. Consistent usage
49. Consistent word and metalanguage element usage
50. Formatting issue
51. Consistent usage
52. Make text more explicit
53. Formatting issue
54. Consistent usage
55. Consistent word and metalanguage element usage; definition of variable inserted
56. Formatting issue
57. Consistent usage
58. Clarification
59. Formatting issue
60. Formatting issue
61. Formatting issue
62. Formatting issue
63. Consistent usage
64. Consistent usage
65. Singular/plural consistency
66. Correction: the caret must be in the value of the expression
67. See 42, consistent singular/plural usage
68. Formatting issue
69. Consistent usage
70. Definition of variable inserted.
71. Correction: the caret is part of the name
72. Correction: the caret must be in the value of the expression
73. Clarification
74. See 42, consistent singular/plural usage
75. Consistent usage
76. Consistent usage
77. See 42
78. See 42
79. Clarification. Users may belong to multiple groups.
80. Clarification. (What “thingy” is being initiated)
81. Formatting issue
82. See 42, consistent singular/plural usage
83. Definition of variables inserted, see 42, consistent singular/plural usage
85. See 42, consistent singular/plural usage
86. Formatting issue
87. Formatting issue
88. Formatting issue
89. Consistent usage
90. Already done in Draft # 15
91. Clarification
92. Formatting issue
93. Clarification
94. Obsolete, taken care of in Draft # 15
95. Obsolete, taken care of in Draft # 15
96. See 42 and “general change” below
97. Correction
98. Correction, see 97
99. Clarification
100. Formatting issue
101. Indefinite article needed
102. Clarification
103. Missing word
104. Make text more explicit

105. Correction: initialization happens only once. The activity described is a normal change of status.
106. Correction, see 97
107. Taken care of in Draft # 15
108. Correction, see 105
109. Correction, see 97
110. Formatting issue
111. Formatting issue
112. Consistent word usage
113. Formatting issue
114. Clarification, mandated by Interpretations Taskgroup
115. Clarification, mandated by Interpretations Taskgroup
116. This paragraph is now referenced from ^$Character
117. Consistent word usage
118. See “General change”
119. Consistent word usage
120. Missing comma
121. Taken care of in Draft # 15
122. Consistent word usage
123. Consistent word usage
124. Missing sentence
125. See “General change”
126. See 13
127. Clarification. Make text more explicit
128. See 13
129. Clarification. Make text more explicit
130 ! 139: Spelling of pi
140: Clarification
141: Clarification, be consistent with 140
142: Clarification.
143, 144: Spelling of pi
145: Typo
146: Typo
147: Clarification, use consistent terminology, insert missing item
148: Insert missing item
149: Insert missing item
150: Formatting issue
151: Typo, no opposing choices intended
152: Typo
153: Clarification
154. Clarification
155. Clarification, use consistent terminology
156. Taken care of in Draft # 15
157: See “General Change”
158: Taken care of in Draft # 15
160: Correction: by definition, an exvar cannot have a parameter list
161: Formatting issue, taken care of in Dradt #16
162: Clarification
163: Consistent usage
164: Correction: by definition, an exvar cannot have a parameter list
165: Taken care of in Draft # 15
166: Typo, taken care of in Draft # 15
167: See “General Change”
168: Consistent word usage, clarification
169: Clarification
170: Word-order made more clear
171: Clarification (see 169)
172: Clarification
173: Consistent usage. Usage like “the FOR” or “the ELSE” should be more formal: “the FOR command” and “the ELSE command”.

174: See 173.
175: Correction: the FOR command cannot have multiple arguments.
176: See 173
177: Word usage: do not use adjectives as substantives.
178: Clarification: the term “scope” needs a frame of reference
179: Typo
180: See 173
181: See 173
182: See 178
183: See 173
184: See 173
185: Clarification: the term “it” has no antecedent
186: See 173
187: Clarification
188: See 173
189: See 173
190: See 173 and 187
191: See 173
192: See 173, clarification
193: See 173
195: Clarification
196: Formatting issue, taken care of in Draft # 15
197: Clarification: remove implementation-specific terms that are not defined in the standard, make word usage consistent
198: Formatting issue, taken care of in Draft # 15
199: Formatting issue, omit repetition
200: Formatting issue, taken care of in Draft # 15
201: Clarification
202: Clarification
203: Formatting issue, taken care of in Draft # 15
204: Clarification
205: Clarification
206: Insert missing verb, see 173
207: See 173
208: See 173
209: See 173
210: See 173
211: See 173
212: See 173
213: See 173
214: Make consistent with internationalization
215: See 173
216: Formatting issue
217: Remove internal contradiction from sentence
218: Make consistent with current number of leftrestricteds
219: Move specific case to location where it is referenced
220: Typo
221: Formatting issue
222: Consistent word usage
223: See 173 (applies three times more than in X11/1999-7)
224: See 173 (applies once more then in X11/1999-7)
225: See 173 (applies once more than in X11/1999-7)
226: See 42
227: See 173
228: See 173
229: See 173
230: See 173
231: See 173
232: Clarification

233: Typo
234: Correction: the specification at hand is not the definition of the metalanguage element
235: Formatting issue
236: See 42
237: Typo, strange “smart quotes”
238: Clarification: quantity should have a unit
239: See 238
240: Taken care of in Draft # 15
241: Taken care of in Draft # 15
242: Taken care of in Draft # 15
243: Remove year-number (any version of X3-64 would apply)
244: Clarification
245: Formatting issue
246: Taken care of in Draft # 15
247: Use appropriate verb: formal language requires “shall”
248: Typo
249: Formatting issue, in addition: typo in the word “fourth”
250: Typo
251: Typo
253: Clarification
254: Clarification
255: Clarification
256: Mostly taken care of in Draft # 15, clarified numbers 9, 94. 95, 101, 102, 103, 104, 105, 106, 107, 108
257: Typo, taken care of in Draft # 15
258: Remove inappropriate entries
259: Formatting issue
260: Formatting issue
261: Formatting issue
262: Remove extraneous entry
263: Additional entries; some already applied in Draft # 15
264: Typo
265: Typo
266: Missing text inserted
267: Missing text inserted
268: Missing text inserted
269: Missing text inserted
270: Missing specifications inserted
271: Clarification
272: Consistent word usage
273: Consistent word usage
274: Remove obsolete clause
275: Insert missing entry
276: Typo
277: Typo
278: Clarification
279: Insert missing character
280: Insert missing character
281: Typo
282: Taken care of in Draft # 15
283 through 338: MUMPS code re-applied from original MDC Type A documents
339: New code inserted

General change:
The terms: future enhancement(s), future expansion(s) and future extension(s) are all replaced by “future extension”, or “reserved for future extensions to the standard” (see 42).

Recommendations that will need discussion and possibly a vote are:

4. Remove curly braces, or remove text within braces as well?
10. Remove brackets
36. More in line with the approved MDC Type A
252: Annex A, table A.1
Recommend to remove the columns labeled "2nd Order" and "3rd Order", and to change the label of the remaining collation-related column from "Collation Table, 1st Order" to "Collation Order".
If this recommendation is followed, also remove the line following the table that states:
Note: 2nd and 3rd order collation values happen to be blank (i.e., not needed) for this Character Set Profile definition; the 1st order collation value happens to be unique across all the characters in this profile.
Note: ISO-8859!1-USA does use three levels to define collation. For some languages more than three levels are needed, so I'm afraid that we cannot use the three-level model as a template for all character set profiles.

Recommendations that will definitely require a vote are:

17. The statement that "the first line of a routine is at level 1" is erroneous: there is no text in the standard that states that the first line of a routine may not have dots in its linestart. However, the target of a DO command must be at level one (see definition of DO command). Recommend to change this sentence to:
The first line of a callable function or subroutine must be at level 1 (see xxx) and ...

19. The current definition continues to cause misunderstandings. The proposed new language is based on the research of Fred Hiltz and Ed de Moel.

32. There needs to be a formal proposal that defines when the errors M21 and M57 occur.

35. Since there is seemingly a disparity between parts of the standard, and this modification would make one possible interpretation consistent throughout the standard, it would seem appropriate to vote that this interpretation is the intended one (i.e. error M14 will occur when the target of a DO command is a line that is not at level 1).

84. A proposal needs to be drafted to specify a model for the LOCK table.

159. Section 1, clause 8.1.6.1
In any context, reference to a particular spelling of label which occurs more than once in a defining occurrence in the given routine will have undefined results.
Should be
In any context, reference to a particular spelling of label which occurs more than once in a defining occurrence in the given routine will not be possible, because the insertion of such a duplicate label will cause an error with ecode = "M57".

194: Section 1, clause 8.2.7
The specification of the HALT command does not say anything about what happens to devices that are OPEN when a job is HALTed.
Is behavior implementation-specific?
Are all OPENed devices CLOSEd?
Do we want to say anything specific here?
[I remember that this issue came up when we were discussing the final version of the 1990 standard. At that time, David Marcus was strongly opposed to including any specific language to the standard in this context. But... as far as I know, his implementation neatly CLOSEs all OPENed devices, so... Have things changed? What do other implementations do?]

340: It is very hard to search the document for occurrences of the name of the language. Recommend to spell the name of the language consistently as M[UMPS].

14. Editorial changes applied to Draft # 15, other than those itemized in X11/1999-7

Reformatted all meta-language elements to be in the same font, and aligned identically.

Removed all unused character styles and paragraph styles from the document.

Put all references to meta-language elements in the same character style.

Spelled all dates in accordance with guidelines from Chicago Style Guide (day in digits, month fully spelled in letters, year fully spelled in digits).

Re-numbered all sections and clauses to be automatically updated when items are inserted or deleted.

Re-coded all cross-references to use “automated” page numbering.

Checked the modifications for all MDC Type A Extensions. Corrected some minor typos. Added some omitted phrases.

Made all texts about the initialization of special variables consistent. Used the text from $STack.
Current text now reads for all svns

7.1.4.10.6 $IO and 7.1.4.10.7 $IOREFERENCE
changed “this OPEN and USE” to “these OPEN and USE commands”

Added several notes in the text of the standard about issues that require resolutions. These notes appear as white characters on a black background. These notes should be removed before final publication.

7.1.6.5.4 $%ARCCOT^MATH
Changed “ARCCOS” to ARCCOT”

In the trigonometry functions, an attempt was made to use the term “in radians” in a consistent fashion. Unfortunately, the result is that about half of the references ended up in the wrong place. For a function like a sine, the parameter is in radians and the function value is a scalar number, whereas for a function like a arcsine, the function value is in radians and the parameter is a scalar number. Reworded all occurrences to be mathematically correct.

7.1.6.2 Library element definitions
changed “M (mandatory)” to “mandatory” (in this version of the standard, the letter M is no longer defined as a specification for a parameter).

Completed the instruction to change | environment | in all meta language definitions to VB environment VB.

Reworded the definition of $%PRODUCE and $%REPLACE to be more similar to other library specifcations (let xxx be the value of ...)

In 7.1.3.5.3 changed “ The node "xxx" is the value of xxx” to “The value of node xxx is equal to the value of xxx” twice.

In 7.1.3.9.1 changed “ASCII/M” to any standardized characterset profile.
(ASCII and M are no longer the only two standardized ones...)

In Annex A, changed “if $CV_j(t)$” to “if there is a *j* such that $CV_j(t)$” twice.

In a number of cases, the quotes were missing in constructions like expr V “abcde”.

In annex G, Clause 3, re-wrote the final sentence to read:
Let *n* be a decimal value. If there is no character assigned for *n* in JIS X0208, then the external

representation of $CHAR(*n*) will be the same as the Japanese space, or $CHAR(8481).

In 7.1.6.4.2, in the description of the function value, “compares” is replaced by “collates” (three times).

Changed the description of M99 to “Invalid operation for socket context”

7.1.3.1.6 Case conversion

Made the metalanguage consistent for the right-hand side of the equal sign: expr V algoref.

7.1.6.4.3 and 7.1.6.4.4

Reworded the text to become (with upper and lower transposed appropriately):

$%UPPER^STRING returns a string that is an edited version of the value of its first parameter, in which all lower-case characters are converted to the corresponding upper-case characters.

If the value of CHARMOD is a namevalue referencing a gvn, then the conversion algorithm used is that specified in ^$GLOBAL for that gvn. If the value of CHARMOD is a charsetexpr, then the conversion algorithm used is that specified in ^$CHARACTER for that character set profile. If CHARMOD is not specified, or the node specified above does not exist, then the conversion algorithm used is that specified as the default for the process (either in ^$JOB or ^$SYSTEM).

If no algorithm is specified in the appropriate ssvn, then the characters a through z are converted to A through Z respectively.

If CHARMOD references a gvn, it must be either of the form ^ name or of the form ^ VB environment VB name.

The sample code of the functions for UPPER, LOWER and PATCODE needs several corrections.

6.3.3.1 Event classes

The format of the list of event classes was brought more in line with other enumerations.
In each section, the start was changed from “These are events that” into “The event class xxx contains events that ...”

6.3.3.4 Synchronous event handling
Changed “... the number of M QUIT commands” to “... the number of QUIT commands”.

8.2.xxx ASTART and ASTOP and AUNBLOCK
Changed “aeventargument” to “ablockargument”.

8.2.xx ESTART
Changed “It is not an error to issue a second ESTART command on the same event classes.” to “It is not an error to issue multiple ESTART commands on the same event class.”

15. X11/1998-19: User-defined system variables
Changed the last three paragraphs to:

Let *r* be the name of the routine being called when a reference is made to a certain user-defined structured system variable, and let *l* be the label at which this routine is called. If the routine does not exist when a reference is made to the user-defined structured system variable, then an error condition occurs with ecode = M97. If the routine exists, but the label does not exist when a reference is made to the user-defined structured system variable, then an error condition occurs with ecode = M13.

Note: names of user-defined structured system variables which differ only in the use of corresponding upper and lower case letters are not equivalent.

Note: Users providing routines to implement user-defined structured system variables are responsible for ensuring that other side-effects (such as a change to $TEST or $DATA values), which would not have taken place, had the reference been to a global variable, do not occur as a result of calling the routine.

16. X11/1998-29: Local variables in ^$JOB.
Changed the term “variable” in the context of this Type A, everywhere to “local variable”.

17. X11/SC13/1998-10: FORMAT^STRING
The corrections from X11/SC13/TG2/1999-1 were applied.

The Type A proposal assumes that certain nodes in ^$SYSTEM and ^$FORMAT are available, but does not contain any instructions to add a definition of these nodes to the standard. The document editor has taken the liberty to add definitions for

^$SYSTEM(system,”FORMAT”,”CS”)
^$SYSTEM(system,”FORMAT”,”DC”)
^$SYSTEM(system,”FORMAT”,”EC”)
^$SYSTEM(system,”FORMAT”,”FS”)
^$SYSTEM(system,”FORMAT”,”FM”)
^$SYSTEM(system,”FORMAT”,”SL”)
^$SYSTEM(system,”FORMAT”,”SR”)

and

^$JOB(job,”FORMAT”,”CS”)
^$JOB(job,”FORMAT”,”DC”)
^$JOB(job,”FORMAT”,”EC”)
^$JOB(job,”FORMAT”,”FS”)
^$JOB(job,”FORMAT”,”FM”)
^$JOB(job,”FORMAT”,”SL”)
^$JOB(job,”FORMAT”,”SR”)

18. X11/SC13/TG3/1998-4: $DEXTRACT and $DPIECE, Data Record Functions
Changed “left-aligned” and “right-aligned” to “left-justified” and “right-justified” respectively.
changed “Although recordfieldvalue is optional” to “Although all elements of the list of recordfieldvalues are optional”
changed “Although recordfieldglvn is optional” to “Although all elements of the list of recordfieldglvns are optional”

19. X11/SC13/TG6/1998-3: $HOROLOG function
An early version of this proposal was already entered into the document. Removed the parts that were not included in the final version of the proposal.

20. X11/SC15/1998-11: Generic Indirection
The examples in the proposal have clearly been modified to support the statement that insertion of cs is considered to be erroneous. The text of the formalism still indicates that such insertions are allowable. The inconsistent text has been removed.

21 Typographical and editorial modifications between draft version 17 and 18

Clause 4.11: (glossary, character) changed {graphic, phonetic} symbol to graphic symbol, phonetic symbol.
Clause 4.24: (glossary, diacritic) italicized term *letter* where appropriate
Clause 7.1.3: (list of ssvns) insert Y[unspecified]
Clause 7.1.3.1.3: (valid name characters for ^$CHARACTER) Make code sample stand out: separate line and “code” font
Clause 7.1.3.1.4: (patcode definition) Change font of code sample to “code” font
Clause 7.1.3.2: (^$DEVICE) change “implementation specific” to “implementation-specific”
Clause 7.1.3.3: (^$EVENT) add quotes around EVENTDEF
Clause 7.1.3.4.1: (Collation algorighm, ^$GLOBAL) Change font of code sample to “code” font
Clause 7.1.3.9.1: (system character set profile, ^$SYSTEM) the abbreviation “etc” appeared without a

terminating period. Given the choice between ending a sentence with multiple periods and spelling the term in full, I chose to spell the term in full.
Clause 7.1.4.2: (definition of intlit) Insert the missing period before the term intlit in the definition of mant.
Clause 7.1.4.9: (exvar) Change the font of the code sample to “code” font
Clause 7.1.4.12: (namevalue) Insert a space between the vertical bar and the ellipsis
Clause 7.1.5.8: ($FNUMBER) Reduced the space between the vertical bars in the definition of fncodatom
Clause 7.1.5.11: ($MUMPS) Adjust the vertical alignment in the definition of noncommasemi
Throughout: replace simple dashes by minus sign (n-spaced dashes)
Clause 7.1.5.23 ($STACK): put quotes around PLACE, MCODE and ECODE
Clause 7.1.6.4.2: ($%COMPARE) replace the term “compares” by “collates” (three times)
Clause 7.1.6.5.18: ($%CLOG) Insert the missing parenthesis in RE (Z)
Clause 7.1.6.6.3: ($%PRODUCE) Replaced “the first I in SPEC” with “the first *e* in SPEC”
Clause 7.2.2: (truthop) Reduced the space between the vertical bars in the definition of truthop
Clause 7.2.3: (pattern match) Reduced the space between the vertical bars in the definition of repcount
Clause 7.2.3: (pattern match) Replaced “if S2" by “if the value of S2"
Clause 8.1: (list of commands) Replaces RL[0AD] by RL[OAD] (zero versus ooh)
Clause 8.1.9: (user defined I/O): Re-adjusted alignment of columns in itemizations
Clause 8.2.25: (OPEN) Changed the font of the definition of mnemonicspace to “normal”
Clause 8.2.29: (RSAVE) Inserted a comma: “... 1 or 11 exists the routine” is now “... 1 or 11 exists, the routine”
Clause 8.2.30: (SET) Changed “...there are no subscripts, is modified” to “...there are no subscripts. The setleft is modified”
Clause 8.2.34: (TSTART) Reduced the space between the vertical bars in the definition of restartargument
Clause 8.2.35: (USE) Added See 8.2.7 for the syntax and interpretation of devn and deviceparameters.
Clause 8.2.37: (WRITE) Changed the font of the definition of tabformat to “normal”
Annex F: removed box-lines, and put headers in grey boxes
Annex I: Insert sample code for COLLATE and COMPARE
Clause 7.1.3.1.1: (input transformation) Changed font for sample code to “code” font and put sample on separate line.
Clause 7.1.3.1.2: (output transformation) Changed font for sample code to “code” font and put sample on separate line.
Clause 7.1.3.1.4: (patcode definition) Removed redundant spaces.
Clause 7.1.3.4.1: (collation algorithm) Removed redundant spaces
Clause 7.1.3.10 (^$Y) Underlined the term ssvn.
Clause 7.1.5.14: ($ORDER) Removed redundant spaces, centered resulting text (twice)
Clause 7.1.6.6.4: ($%FORMAT^STRING) Changes simgular to singular
Clause 7.1.6.6.4: ($%FORMAT^STRING) Filled in page numbers for references to ^$JOB and ^$SYSTEM
Annex B: (error M103) Changed “Inval;id” to “Invalid”
Throughout: spelled names of functions, special variables and structured system variables in full, rather than abbreviated

22. Corrections to error codes 95 and 99

M95 was: “Imaginary Number”, is now: “Result value has non-zero imaginary part”
M99 was “Invalid operation for context” is now “Invalid operation for socket context”

23. Suggestions for error codes 9, 21, 28, 39, 46, 47 and 113

Error M21 (name occurs more than once in a formallist) should probably occur while executing an RSAVE command.
For M21, the text is “Algorithm specification invalid”. Suggest to change that to:
Multiple occurrences of same formal parameter name.

Clause 7.1.6.5.24: When an attempt is made to calculate the hyperbolic cotangent of 0 (zero), should the error be M9 (division by zero) or M28 (invalid parametervalue)?

24. Definitions that are not part of X11.1 but appear in X11.6

Clause 8.1.12: the metalanguage symbol einforef is defined in X11.6, but referenced here in X11.1. Suggest to specify the definition as a reference to X11.6 with a foot-note that details the definition in X11.6.

25. Remaining specifications from sockets binding

The specifications from the “Sockets Binding” were included only partially in Draft Version 14. The remaining parts are inserted in this version.

After clause 7.1.3.2.3: fill in additional specifications
Annex H: fill in additional specifications

26. Other changes and additions between Version 17 and Version 18

Clause 7.1.3.1.5: insert specification of nodes for LOWER and UPPER conversion (p489vv)
Clause 7.1.3.9.2: insert specification for ^$Y (page 481vv)
Clause 8.1.6.1: this clause states that a result is not defined, whereas elsewhere the result is defined to be an error with code M57
Clause 8.2.30: [SET], definition of setqsub: should the first parameter be glvn, a namevalue or a glvn V namevalue? Added a suggestion to make it glvn V namevalue.
Annex I: should the functions LOWER and UPPER be part of STRING or of CHARACTER?
Clause 8.2.30 (SET): added suggestion to change glvn in specification of SET $QSUBSCRIPT to glvn V namevalue.
Foreword: included the results of the vote on 19 September 1998.
Clause 7.1.3.1.5: (collation) Added suggestion to insert an example of how the algoref would be used.
Clause 7.1.3.1.6: (upper and lower case conversion) Added suggestions to insert examples of how the algorefs would be used.
Clause 7.1.5.21: ($REVERSE) Added suggestion to replace algorithm by one that fits better with the portability requirements.
Clause 7.1.6.4.3: ($%LOWER^STRING) Suggest to rename to $%LOWER^CHARACTER
Clause 7.1.6.4.4: ($%PATCODE^STRING) Suggest to rename to $%PATCODE^CHARACTER
Clause 7.1.6.4.5: ($%UPPER^STRING) Suggest to rename to $%UPPER^CHARACTER
Clause 7.2.2.1: (Relational operators) The term “dual” was introduced into the standard in a day and age that all relational operators were single characters, and indicated that the described “combined” operators would have two characters. Suggest to replace the word “dual” with “multi-character”.
Clause 8.2.12: (ETRIGGER) Added suggestion to include definitions for einforef and espec from X11.6 as a footnote.

Foreword

(This Foreword is not part of American National Standard MDC X11.1-Millennium.)

M[UMPS] is a high-level interactive computer programming language developed for use in complex data handling operations. It is also known as MUMPS, an acronym for Massachusetts General Hospital Utility Multi-Programming System. The MUMPS Development Committee has accepted responsibility for creation and maintenance of the language since early 1973. The first ANSI approved standard was approved 15 September 1977 via the canvass method. The standard was revised and approved again on 15 November 1984, on 11 November 1990, and again on 8 December 1995. Subsequently, the MUMPS Development Committee has met several times annually to consider revisions to the standard.

On 19 September 1998, the MUMPS Development Committee passed a motion to include all extensions that were approved at that time into the Draft Millennium Standard, and to present the resulting document for approval as an American National Standard through the canvass process.

Document preparation was performed by the MUMPS Development Committee. Suggestions for improvement of this standard are welcome. They should be submitted to the MUMPS Development Committee, c/o MDC Secretariat, 800 Nelson Street, Rockville, MD 20850-2051.

Introduction

Section 1 consists of nine clauses that describe the M[UMPS] language. Clause 1 describes the metalanguage used in the remainder of Section 1 for the static syntax. The remaining clauses describe the static syntax and overall semantics of the language. The distinction between "static" and "dynamic" syntax is as follows. The static syntax describes the sequence of characters in a routine as it appears on a tape in routine interchange or on a listing. The dynamic syntax describes the sequence of characters that would be encountered by an interpreter during execution of the routine. (There is no requirement that M[UMPS] actually be interpreted). The dynamic syntax takes into account transfers of control and values produced by indirection.

1 . Scope

This standard describes the M[UMPS] programming language.

2 . Normative References

The following standard(s) contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standard(s) indicated below. Members of ANSI maintain registers of the currently valid standards.

ANSI X3.135!1992 Information Systems - Database Language - SQL
ANSI X3.4-1990 (ASCII Character Set)
ANSI X3.64!1979 R1990 (Additional controls for use with American National Standard Code for Information Interchange)
ANSI X11.2!1995, Open MUMPS Interconnect
ANSI X11.3!1994, MUMPS - GKS Binding
ANSI X11.4!1995, MUMPS - X-Window™ Binding
ANSI X11.6!1995 M[UMPS] Windowing API

3 . Conformance

3.1 Implementations

A *conforming implementation* shall

a) correctly execute all programs conforming to both the Standard and the implementation defined features of the implementation

b) reject all code that contains errors, where such error detection is required by the Standard

c) be accompanied by a document which provides a definition of all implementation-defined features and a conformance statement of the form:

> "*xxx* version *v* conforms to X11.1-*yyyy* with the following exceptions:
> ...
> Supported Character Set Profiles are ...
> Uniqueness of the values of $SYSTEM is guaranteed by ...
> The minimum amount of local variable storage for a job is guaranteed to be ...
> The depth of event queues is...
> The number of timer events is...
> The resolution of timers is..."

where the exceptions are those components of the implementation which violate this Standard or for which minimum values are given that are less than those defined in Section 2.

An *MDC conforming implementation* shall be a conforming implementation except that the conforming document shall be this Standard together with any such current MDC documents that the vendor chooses to implement. The conformance statement shall be of the form:

> "*xxx* version *v* conforms to X11.1-*yyyy*, as modified by the following MDC documents:
> *ddd* (MDC status *m*)

with the following exceptions:
...
> Supported Character Set Profiles are ...
> Uniqueness of the values of $SYSTEM is guaranteed by ...
> The depth of event queues is...
> The number of timer events is...
> The resolution of timers is..."

An *MDC strictly conforming implementation* is an MDC conforming implementation whose MDC modification documents only have MDC Type A status and which has no exceptions.

A <*National Body*> *... implementation* is an implementation conforming to one of the above options in which the requirements of Section 2 are replaced by the <National Body> requirements and other extensions required by the <National Body> are implemented.

An implementation may claim more than one level of conformance if it provides a switch by which the user is able to select the conformance level.

3.2 . Programs

A *strictly conforming program* shall use only the constructs specified in Section 1 of this standard, shall not exceed the limits and restrictions specified in Section 2 of the Standard and shall not depend on extensions of an implementation or implementation-dependent features.

A *strictly conforming non-ASCII program* is a strictly conforming program, except that the restrictions to the ASCII character set in Section 2 are removed.

A *strictly conforming <National Body> program* is a strictly conforming program, except that the restrictions in Section 2 are replaced by those specified by the <National Body> and any extensions specified by the <National Body> may be used.

A *conforming program* is one that is acceptable to a conforming implementation.

4 . Definitions

For the purposes of this standard, the following definitions apply.

4.1 argument (of a command): M[UMPS] command words are verbs. Their arguments are the objects on which they act.
4.2 array: M[UMPS] arrays, unlike those of most other computer languages, are trees of unlimited depth and breadth. Every node may optionally contain a value and may also have zero or more descendant nodes. The name of a subscripted variable refers to the root, and the *n*th subscript refers to a node on the nth level. Arrays vary in size as their nodes are set and killed. See also *scalar*, *subscript*.
4.3 atom: a singular, most-basic element of a construction. For example, some atoms in an expression are names of variables and functions, numbers, and string literals.
4.4 block: one or more lines of code within a routine that execute in line as a unit. The argumentless DO command introduces a block, and each of its lines begins with one or more periods. Blocks may be nested. See also *level*.
4.5 call by name: a calling program names an actual parameter and passes its value to an object's service. Limited to a single value, that is, the value of a scalar variable or of one node in an array. See also *call by reference*, *call by value*.
4.6 call by reference: a calling program passes to a subroutine or function a reference to its actual parameter. If the called subroutine, function, or object's service changes the value of its formal parameter, the change affects the actual parameter as well. Limited to unsubscripted names of local variables, either scalar or array. See also *call by name*, *call by value*.
4.7 call by value: a calling program passes the value of its actual parameter to a subroutine, function, or object's service. Limited to a single value, that is, the value of a scalar variable or of one node in an array. See also *call by name*, *call by reference*.
4.8 call: a procedural process of transferring execution control to a **callee** by a **caller**.
4.9 callee: the recipient of a **call**.
4.10 caller: the originator of a **call**.
4.11 character: (1) a member of a set of elements used for the organization, control, or representation of data. (2) a character is a simple or composite graphic symbol belonging to a conventional set of symbols. There are alphabetic characters, numerical characters (arabic and roman), diacritic characters (for example $ / ! '), punctuation characters (for example . , ; : ! ?), and specific other characters (for example § $ % & { #). The following synonyms should be avoided: graphic symbol, phonetic symbol, sign, mark, note, cipher, whether or not these are used in conjunction with terms like phonetic and graphic..
4.12 combining character: a member of an identified subset of the coded character set of ISO/IEC 10646 intended for combination with the preceding non-combining graphic character, or with a sequence of combining characters preceded by a non-combining character (see also *composite sequence*). NOTE - ISO/IEC 10646 specifies several subset collections which include combining characters.
4.13 command: a command word (a verb), an optional conditional expression, and zero or more arguments. Commands initiate all actions in M[UMPS].
4.14 composite sequence: a sequence of graphic characters consisting of a non-combining character followed by one or more combining characters (see also *combining characters*). NOTES - 1) A graphic symbol for a composite sequence generally consists of the combination of the graphic symbols of each character in the sequence. 2) A composite sequence is not a character and therefore is not a member of the repertoire any character set.
4.15 computationally equivalent: the result of a procedure is the same as if the code provided were executed by a M[UMPS] program without error. However, there is no implication that executing the code provided is the method by which the result is achieved.
4.16 concatenation: the act or result of joining two strings together to make one string.
4.17 conditional expression: guards a command (sometimes an argument of a command). Only if the expression's value is true does the command execute (or the argument). See also *truthvalue*.
4.18 contains: a relational operator that tests whether one string is a substring of another.
4.19 data-cell: in the formal model of M[UMPS] execution. It contains the value and subscripts (if any) of a variable, but not the name of the variable. Any number of variable names may point to a data-cell due to parameters passed by reference. See also *name-table*, *value-table*.
4.20 default property: the default state of an

object; a property to which an OREF evaluates if used in a property reference that doesn't name a specific property. See also *default state*, *OREF*, *property*.
4.21 default state: the state that is assumed when no state has been explicitly specified.
4.22 descriptor: uniquely defines an element. It comprises various characteristics of the element that distinguish the element from all other similar elements.
4.23 device-dependent: that which depends on the device in question.
4.24 diacritic: character which is not a letter of the latin alphabet and which is placed over, under, or through a *letter* or combination of *letters* indicating a semantic or phonetic value different from that given the unmarked or otherwise marked *letter*. A *letter* with a diacritic is a composite character. NOTE - The points of german “Umlaut"-character should be regarded as diacritic. [also called a *diacritical mark*]
4.25 diacritic mark: an attribute of a character, written either above or below that character, applied to denote a phonetically or linguistically different letter.
4.26 digit: a graphic character used to represent the numeric value, or part thereof, of a number. Examples: decimal digits, hexadecimal digits.
4.27 empty: an entity that contains nothing. For example, an empty string contains no characters; it exists but has zero length. See also *null string*, *NULL character*.
4.28 environment: a set of distinct names. For example, in one global variable environment all global variables have distinct names. Similar to a directory in many operating systems.
4.29 evaluate: to derive a value.
4.30 execute: to perform the operations specified by the commands of the language.
4.31 “executing” a namevalue: a namevalue is “executed” when it is used in an indirect reference (i.e., @Ref) or subscripted indirectness (i.e., @Ref@(3)).
4.32 extract: to retrieve part of a value, typically contiguous characters from a string.
4.33 extrinsic: a function or variable defined and created by M[UMPS] code, distinct from the primitive functions or special variables of the language. See also *intrinsic*.
4.34 follow: to come after according to some ordering sequence. See also *sorts after*.
4.35 function: a value-producing subroutine whose value is determined by its parameters. Intrinsic functions are defined elements of the language, while extrinsic functions are programmed in M[UMPS].
4.36 global variable: a scalar or array variable that is public, available to more than one job, and persistent, outliving the job. See also *local variable*.
4.37 GMT: Greenwich Mean Time.
4.38 graphic: a visible character (as opposed to most control characters).
4.39 graphic character: a character, other than a control function, that has a visual representation normally handwritten, printed, or displayed.
4.40 hidden: unseen. The NEW command hides local variables. Also pertains to unseen elements invoked to define the operation of some commands and functions.
4.41 intrinsic: a primitive function or variable defined by the language standard as opposed to one defined by M[UMPS] code. See also *extrinsic*.
4.42 job: a single operating system process running an M[UMPS] program.
4.43 label: identifies a line of code.
4.44 letter: (1) a letter (or alphabetic character) is a character that is an individual unspecific basic unit of an alphabet, irrespective of the shape and any graphical realization on a medium. In some alphabets, a letter can be specified as a *small letter* or *capital letter*. (2) a graphic character used for writing natural language, normally representing a sound of the language.
4.45 level: the depth of nesting of a block of code lines. The first line of a routine is usually at level 1 and successively nested blocks are at levels 2, 3, . . . Formally, the level of a line is one plus li. Visually, li periods follow the label (if any) and precede the body of the line. See also *block*.
4.46 library: a collection of library elements, with unique names, which are referenced using a single library name. A library is defined as being either mandatory or optional.
4.47 library element: an individual function that is separately defined and accessible from an M[UMPS] process using the library reference syntax.
4.48 ligature: (1) a composite character joining two or more letters. There are ligatures that are conventionalized units of national variants of alphabets, and ligatures that are caused by the font used in a document. Maybe the first ones should be named *ligature characters*, the last ones *ligature font elements*. [Language dependent. Only *ligature characters* are taken into consideration] (2) two or more letters written together. The resulting symbol is in some cases considered equivalent with the originating letters, in some cases it is considered a separate entity.
4.49 local variable: a scalar or array variable that is private to one job, not available to other jobs, and disappears when the job terminates.

See also *global variable*.
4.50 lock: to claim or obtain exclusive access to a resource.
4.51 M[UMPS] Standard Library: all libraries and library elements defined within the M[UMPS] Standard, whether mandatory or optional.
4.52 mapping: the logical association or substitution of one element for another.
4.53 map: the act of mapping.
4.54 metalanguage: underlined terms used in the formal description of the M[UMPS] language.
4.55 method: a service that represents the behavior that may be requested of an object. See also *object*, *property*, *service*.
4.56 modulo: an arithmetic operator that produces the remainder after division of one operand by another. There are many interpretations of how this operation is performed in the general computing field. M[UMPS] explicitly defines the result of this computation.

Note from X11/1999-7:
replace the definition by:
... produces the result of a mapping of one operand onto a subspace defined by the other operand. For a mathematical definition see D.E. Knuth, The Art of Computer Programming, Volume I, Fundamental Algorithms (or any other book about Abelian Group theory).
Note – Modulo is not the same as remainder; M[UMPS] does not have a remainder operator.

4.57 multidimensional: used in reference to arrays to indicate that the array can have more than one dimension.
4.58 MVAL: the type of any data value that may be represented as a string of variable length. Arithmetic operations interpret strings as numbers, and logical operations further interpret the numbers as true or false. See also *OREF*, *truthvalue*, *type*.
4.59 naked: a shorthand reference to one level of the tree forming a global array variable. The full reference is defined dynamically.
4.60 name-table: in the formal model of M[UMPS] execution, a set of variable names and their pointers to data-cells.
4.61 negative: a numeric value less than zero. Zero is neither negative nor positive.
4.62 node: one element of the tree forming an array. It may have a value and it may have descendants.
4.63 NULL character: the character that is internally coded as code number 0 (zero). A string may contain any number of occurrences of this character (up to the maximum string length). A string consisting of one NULL character has a length of 1 (one).
4.64 null string: 1) a string consisting of 1 (one) NULL character; 2) a string consisting of 0 (zero) characters.
4.65 object: an identifiable, encapsulated software entity whose state and behavior can only be observed or changed by use of its services. An object is considered as a whole in relation to other entities, and is identified by a value of data type OREF. See also *OREF*, *service*.
4.66 ordering: bringing strings of characters into a well-defined sequence using a string comparison specification.
4.67 OREF: an object reference. A value of data type OREF is a reference to an object that uniquely identifies that object. OREFs have no literal representation. Under most circumstances, values of data type OREF are coerced into values of type MVAL based on the value of the default property of the object identified by the value of data type OREF. See also *default property*, *object*, *type*.
4.68 own: to have exclusive access to a resource. In M[UMPS] this pertains to devices and locks.
4.69 parameter: a qualifier of a command that modifies its behavior (for example by imposing a time out), or augments its argument (for example by setting characteristics of a device). Some parameters are expressions, and some have the form keyword=value. See also *argument*.
4.70 parameter (of a function, subroutine, or object's service): The calling program provides actual parameters. In a called function or subroutine formal parameters relate by position to the caller's actual arguments. In a called object's service formal parameters can relate by position or name to the caller's actual arguments. See also *call by name*, *call by reference*, *call by value*, *parameter passing*.
4.71 parameter passing: this alliterative phrase refers to the association of actual parameters with formal parameters when calling a subroutine, function, or object's service.
4.72 partition: the random access memory in which a job runs.
4.73 piece: a part of a string, a sub-string delimited by chosen characters.
4.74 pointer: indirection allows one M[UMPS] variable to refer, or point to, another variable or the argument of a command.
4.75 portable: M[UMPS] code that conforms to the portability section of the standard.
4.76 positive: a numeric value greater than zero. Zero is neither negative nor positive.
4.77 post-conditional: see *conditional expression*.

4.78 primitives: the basic elements of the language.
4.79 process-stack: in the formal model of M[UMPS] execution, a push-down stack that controls the execution flow and scope of variables.
4.80 property: a service that represents the external view of some of an object's data. An object may have a default property. See also *default property*, *method*, *object*, *service*.
4.81 relational: pertaining to operators that compare the values of their operands.
4.82 scalar: single-valued, without descendants. See also *array*.
4.83 scope (of a command): the range of other commands affected by the command, as in loop control, block structure, and conditional execution.
4.84 scope (of a local variable): the range of commands for which the variable is visible, from its creation to its deletion, or from its appearance in a NEW command to the end of the subroutine, function, or block. Scope is not textual, but dynamic, controlled by the flow of execution.
4.85 service: a body of code associated with objects. Services are the only mechanism through which the state of an object may be altered. An object's services include methods and properties. See also *method*, *object*, *property*.
4.86 sorts after: to come after according to an ordering sequence that is based on a collating algorithm. See also *follows*.
4.87 subscript: an expression whose value specifies one node of an array. Its value may be an integer, a floating point number, or any string. Subscripts are sparse, that is, only those that have been defined appear in the array. See also *array*, *scalar*.
4.88 trails: "*A* trails *B*" means that (" "_*A*)]] (" "_*B*) in the appropriate collation sequence; if not specified, it refers to the sequence used for local variables.
4.89 truthvalue: the value of an expression considered as a logical value. When considered as a numeric value, non-zero is true, and zero is false.
4.90 tuple: a sequence of a predetermined number of descriptors (usually a name and a series of subscripts) that identifies a member of a set.
4.91 type: M[UMPS] recognizes only two data types, the reference to an object, or OREF, and the string of variable length, or MVAL. See also *MVAL*, *OREF*.
4.92 UCT: Universal Coordinated Time.
4.93 unbound: in the formal model of M[UMPS] execution, the disassociation of a variable's name from its value.
4.94 undefined: pertaining to a variable that is not visible to a command.
4.95 unsubscripted: see *scalar*.
4.96 value-denoting: representing or having a value.
4.97 value-table: in the formal model of M[UMPS] execution, a set of data-cells.
4.98 variable: M[UMPS] variables may be local or global, scalar or array.
4.99 write-once: a property of an ssvn descriptive of the ability of a M[UMPS] routine to assign a value to it if and only if it does not currently have a $DATA value of 1 or 11.

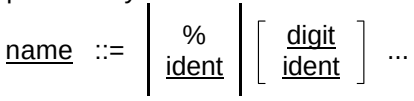
5 . Metalanguage Description

The primitives of the metalanguage are the ASCII characters. The metalanguage operators are defined as follows:

Operator	**Meaning**
::=	definition
[ ]	option
| |	grouping
...	optional indefinite repetition
L	list
V	value
OB	open bracket
CB	close bracket
SP	space
VB	vertical bar

The following visible representations of ASCII characters required in the defined syntactic objects are used: SP (space), CR (carriage-return), LF (line-feed), FF (form-feed), and VB (vertical bar). Also, where necessary to avoid confusion with the “option” metalanguage operator, OB is used to represent the open bracket character ([ ) and CB is used to represent the close bracket character ( ]).

In general, defined syntactic objects will have designators which are underlined names spelled with lower case letters, e.g., name, expr, et cetera. Concatenation of syntactic objects is expressed by horizontal juxtaposition, choice is expressed by vertical juxtaposition. The ::= symbol denotes a syntactic definition. An optional element is enclosed in square brackets [ ], and three dots ... denote that the previous element is optionally repeated any number of times. The definition of name, for example, is written:



The vertical bars are used to group elements or to make a choice of elements more readable.

Special care is taken to avoid any danger of confusing the square brackets in the metalanguage with the ASCII graphics OB and CB. Normally, the square brackets will stand for the metalanguage symbols.

The unary metalanguage operator L denotes a list of one or more occurrences of the syntactic object immediately to its right, with one comma between each pair of occurrences. Thus,

L name

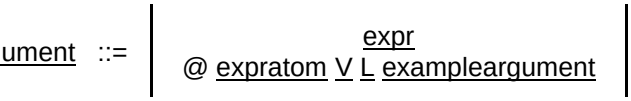
is equivalent to

name [, name] ...

The binary metalanguage operator V places the constraint on the syntactic object to its left that it must have a value which satisfies the syntax of the syntactic object to its right. For example, one might define the syntax of a hypothetical EXAMPLE command with its argument list by

examplecommand ::= EXAMPLE SP L exampleargument

where



This example states: after evaluation of indirection, the command argument list consists of any number of

exprs separated by commas. In the static syntax (i.e., prior to evaluation of indirection), occurrences of @ expratom, that evaluate to valid arguments for the command EXAMPLE, may stand in place of non-overlapping sublists of command arguments. Usually, the text accompanying a syntax description incorporating indirection will describe the syntax after all occurrences of indirection have been evaluated.

6 . Routine routine

The routine is a string made up of the following symbols:

> The graphic, including the space character represented as SP, and also,
> the carriage-return character represented as CR,
> the line-feed character represented as LF,
> the form-feed character represented as FF.

Each routine begins with its routinehead, which contains the identifying routinename. The routinehead is followed by the routinebody, which contains the code to be executed. The routinehead is not part of the executed code.

routine ::= routinehead routinebody

6.1 Routine head routinehead

routinehead ::= routinename eol

routinename ::= name

name ::= | % ident | [digit / ident] ... |

digit ::= The ASCII/M codes 48-57 (characters '0' - '9')

graphic ::= Those characters in the current charset which are not control characters (i.e. do not match the patcode 1C)

ident ::= The ASCII/M codes 65-90 and 97!122 ('A'-'Z' and 'a'-'z') are ident characters, all other characters in the range 0!127 are not ident characters. Additional characters, with codes greater than 127, may be defined as ident through the algorithm specified in ^$CHARACTER(charsetexpr,"IDENT")

eol ::= CR LF

Editor's note:
The character sets ASCII and M are no longer the only standardized ones. Propose to change "the ASCII/M codes xxx" above to "the characters with codes xxx in charset ASCII (and derived character sets)"

names differing only in the use of corresponding upper and lower case letters are not equivalent.

6.2 Routine body routinebody

The routinebody is a sequence of lines terminated by an eor. Each line starts with one ls which may be preceded by an optional label and formallist. The ls is followed by zero or more li (level-indicator) which are followed by zero or more commands and a terminating eol. If there is a comment it is separated from the last command of a line by one or more spaces.

routinebody ::= line ... eor

line ::= | levelline / formalline |

eor ::= CR FF

6.2.1 Level line levelline

A levelline is a line that does not contain a formallist. A levelline may have a LEVEL greater than one. The LEVEL of a line is the number plus one of li. Subclause 6.3 (Routine Execution) describes the effect a line's LEVEL has on execution.

levelline ::= [label] ls [li] ... linebody

li ::= | . [SP] | ...

6.2.2 Formal line formalline

A formalline contains both a label and a formallist which is a (possibly empty) list of variable names. These names may contain data passed to this subroutine (see [8.1.7.1.1] 8.1.7 Parameter passing). A formallist shall only be present on a line whose LEVEL is one, i.e., does not contain an li.

formalline ::= label formallist ls linebody

formallist ::= ([L name])

If any name is present more than once in the same formallist an error condition occurs with ecode="M21".

Editor's note:
The standard does not yet define the exact event that would trigger error M21.
This error should most likely be triggered by the execution of a RSAVE command.

6.2.3 Label label

Each occurrence of a label to the left of ls in a line is called a *defining occurrence* of label. An error condition occurs with ecode = "M57" if there are two or more defining occurrences of label with the same spelling in one routinebody.

Editor's note:
The standard does not yet define the exact event that would trigger error M57.
This error should most likely be triggered by the execution of a RSAVE command.

label ::= | name
intlit |

6.2.4 Label separator ls

A label separator (ls) precedes the linebody of each line. A ls consists of one or more spaces. The flexible number of spaces allows programmers to enhance the readability of their programs.

ls ::= SP ...

6.2.5 Line body linebody

The linebody consists of an optional sequence of commands and an optional comment. Note that the comment always comes after any commands in the line (see 8.1.2 for more about comments). Individual commands are separated by one or more spaces (see 8.1.1 for more about spaces in commands). The end of the line is terminated by a CR LF character sequence.

linebody ::= [commands [cs [comment]] | [commands cs] extsyntax | comment] eol

commands ::= command [cs command] ...

cs ::= SP ...

comment ::= ; [graphic] ...

The use of the extsyntax form is allowed only within the context of an embedded M[UMPS] program (see 6.4 Embedded programs).

6.3 Routine execution

Routines are executed in a sequence of blocks. Each block is dynamically defined and is invoked by an instance of an argumentless DO command, a JOB command (in the case of a new process), a doargument, an exfunc, or an exvar. Each block consists of a set of lines that all have the same LEVEL; the block begins with the line reference implied by the DO, JOB, exfunc, or exvar and ends with an implicit or explicit QUIT command. If no label is specified in the doargument, jobargument, exfunc, or exvar, the first line of the routinebody is used. The *execution level* is defined as the LEVEL of the line currently being executed. Lines which have a LEVEL greater than the current execution level are ignored, i.e., not executed.
Note from X11/1999-7: change the end of the this sentence to:
... are usually ignored (i.e. not executed), except when the line in question is reached as the result of an argumented DO command, in which case an error will occur with ecode = “M14".)

An implicit QUIT command is executed when a line with a LEVEL less than the current execution level or the eor is encountered, thus terminating this block (see 8.2.26 for a description of the actions of QUIT). The initial LEVEL for a process is one. The argumentless DO command increases the execution level by one. (See also the DO command and GOTO command).

Within a given block execution proceeds sequentially from line to line in top to bottom order. Within a line, execution begins at the leftmost command and proceeds left to right from command to command. Routine flow commands DO, ELSE, FOR, GOTO, IF, QUIT, TRESTART, XECUTE, exfunc and exvar extrinsic functions and special variables, provide exception to this execution flow. (See also 6.3.2 Error Processing.) In general, each command's argument is evaluated in a left-to-right order, except as explicitly noted elsewhere in this document.

6.3.1 Generic Indirection

If the evaluation of a command or any of the arguments of a command encounters an indirect expression of the form @expritem which cannot be resolved using the syntax or metatalanguage defined for the command, if appropriate, the @ and expritem are replaced with the value returned by the expritem and the result is interpreted again as if it were part of the original command or linebody. If this replacement results in a syntax which does not match the definition of a routine, an error condition occurs with ecode = “S0”.

The replacement with the value of expritem may not result in the insertion of any of the elements cs, eol, line or eor into the routine.

Editor's note:
What is the error code if any of these elements happens to be inserted. Will there be one error code for all of these insertions, or will each element have its own error code?

Recommend to add:
... If any of these elements is inserted as the result of indirection, an error occurs with ecode = "M113".
Add to Annex B:
M113 Invalid separator inserted

6.3.2 Transaction processing

A TRANSACTION is the execution of a sequence of commands that begins with a TSTART and ends with either a TCOMMIT or a TROLLBACK, and that is not within the scope of any other TRANSACTION. A TRANSACTION may be restartable, serializable, or both, depending on parameters specified in the TSTART that initiates the TRANSACTION. (See 8.2.34 TSTART.) These properties affect execution of the TRANSACTION as described below.

TSTART adds one to the intrinsic special variable $TLEVEL, which is initialized to zero when a process begins execution. TCOMMIT subtracts one from $TLEVEL if $TLEVEL is greater than zero. TROLLBACK sets $TLEVEL to zero. A process is within a TRANSACTION whenever its $TLEVEL value is greater than zero. A process is not within a TRANSACTION whenever its $TLEVEL value is zero.

If, as a result of a TCOMMIT, $TLEVEL would become zero, an attempt is made to COMMIT the TRANSACTION. A COMMIT causes the global variable modifications made within the TRANSACTION to become durable and accessible to other processes.

A ROLLBACK is performed if, within a TRANSACTION, either a TROLLBACK or a HALT command is executed. A ROLLBACK rescinds all global variable modifications performed within the scope of the TRANSACTION, removes any nrefs from the LOCK-LIST that were not included in the LOCK-LIST when the TRANSACTION started (i.e. when $TLEVEL changed from zero to one), and removes any RESTART CONTEXT-STRUCTUREs for both the TRANSACTION linked list and the PROCESS-STACK linked list, discarding the CONTEXT-STRUCTUREs. M[UMPS] errors do not cause an implicit ROLLBACK. (See the LOCK command for definitions of nref and LOCK-LIST.)

Global variable modifications carried out by commands executed within a TRANSACTION are subject to the following rules:

a. A process that is outside of a TRANSACTION cannot access the global variable modifications made within a TRANSACTION until that TRANSACTION has been COMMITted.

b. A process that is inside a TRANSACTION is not explicitly excluded from accessing modifications made by other processes. However, a process cannot COMMIT a TRANSACTION that has accessed the global variable modifications of any other uncommitted TRANSACTION before that other TRANSACTION has been committed.

c. If the transparameters within the argument to the TSTART initiating the TRANSACTION specifies serializability, then all global variable modifications performed by the TRANSACTION and all other concurrently executing TRANSACTIONs must be equivalent to some serial, non-overlapping execution of those TRANSACTIONs.

If it has been determined that a TRANSACTION in progress either cannot or is unlikely to conform to the above-stated rules, then the TRANSACTION implicitly RESTARTs. In addition, the TRESTART command explicitly causes the TRANSACTION to RESTART.

The actions of a RESTART depend on whether it is restartable. A TRANSACTION is restartable if the initiating TSTART specifies a restartargument. (See 8.2.34 TSTART.) A RESTART of a restartable TRANSACTION causes execution to resume with the initial TSTART. A RESTART of a non-restartable

TRANSACTION ends in an error (ecode = "M27").

The following discussion uses terms defined in the Variable Handling (see 7.1.2.2) and Process-Stack (see 7.1.2.3) models and, like those subclauses, does not imply a required implementation technique. Execution of a RESTART occurs as follows:

a. The frame at the top of the PROCESS-STACK is examined. If the frame's linked list of CONTEXT-STRUCTUREs contains entries, they are processed in last-in-first-out order from their creation. If the CONTEXT-STRUCTURE is exclusive, all entries in the currently active local variable NAME-TABLE are pointed to empty DATA-CELLs. In all cases, the CONTEXT-STRUCTURE NAME-TABLEs are copied to the currently active NAME-TABLEs. For each RESTART CONTEXT-STRUCTURE, $TLEVEL is decremented by one until $TLEVEL reaches 0 (zero) or the list is exhausted. If $TLEVEL does not reach 0 (zero), then:

    1. if the frame contains formallist information, it is processed as described by step d in the description of the QUIT command (see 8.2.26).

    2. the frame is removed and step a repeats.

b. $TEST and the naked indicator are restored from the CONTEXT-STRUCTURE that triggered $TLEVEL to reach 0 (zero).

c. A ROLLBACK is performed. If the TRANSACTION is not restartable, RESTART terminates and an error condition occurs with ecode = "M27"

d. $TRESTART is incremented by 1. RESTART terminates and execution continues with the initial TSTART, which includes re-evaluating postcond, if any, and tstartargument, if any.

6.3.3 Error processing

Error trapping provides a mechanism by which a process can execute specifiable commands in the event that $ECODE becomes non-empty. The following facilities are provided:

The $ETRAP special variable may be set to either the empty string or to code to be invoked when $ECODE becomes non-empty. Stacking of the contents of $ETRAP is performed via the NEW command.

$ECODE provides information describing existing error conditions. $ECODE is a comma-surrounded list of conditions.

The $STACK function and $STACK variable provide stack related information.

$ESTACK counts stack levels since $ESTACK was last NEWed.

When an error condition is detected, the information about the error is appended to the current value of $ECODE and to $STACK($STACK,“ECODE”). If appending to $ECODE or $STACK($STACK,“ECODE”) would exceed an implementation’s maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,“ECODE”) to discard. The value of $ECODE may also be replaced via the SET command.

An Error Processing transfer of control consists of terminating the current command and processing in the scope of *any* active FOR commands and indirection; and second, explicitly resuming execution at the same LEVEL with two lines where the body of the first line is the value of $ETRAP and the body of the second line is:

```
QUIT:$QUIT "" QUIT
```

The two lines are:

```
ls [li] x eol
ls [li] QUIT:$QUIT "" QUIT  eol
```

Where li represents the line level at the time of the transfer of control and *x* represents the value of $ETRAP.

For purposes of this transfer each command argument is considered to have its own commandword (see 8.1 General command rules).

When an error condition is detected, the information about the error is appended to the current value of $ECODE and to $STACK($STACK,"ECODE"). The value of $ECODE may also be replaced via the SET command.

An Error Processing transfer of control is performed when:

a. The value of $ECODE is updated to a non-empty value. This occurs when an error condition is detected or may be forced via the SET or ASSIGN command.

b. $ECODE is not the empty string and a QUIT command removes a PROCESS-STACK level at which $STACK($STACK,"ECODE") would return a non-empty string, and, at the new PROCESS-STACK level, $STACK($STACK,"ECODE") would return an empty string (in other words, when a QUIT takes the process from a frame in which an error occurred to a frame where no error has occurred).

When in the context of error processing (i.e., $STACK($STACK,"ECODE") returns a non-empty string) a new error condition occurs (i.e., the value of $ECODE changes to a different non-empty string), the following actions are performed:

a. It associates the information about the failure as if it were associated with the frame identified by $STACK+1.

b. The following commands are implicitly incorporated into the current execution environment immediately preceding the next command in the normal execution sequence:

```
TROLLBACK:$TLEVEL  QUIT:$QUIT "" QUIT  ;
```

6.3.4 Event processing

Event processing provides a mechanism by which a process can execute specifiable commands in response to some occurrence outside the normal program flow. Event processing can be done using either a synchronous model or an asynchronous model. Synchronous event processing is enabled by issuing the ESTART command, and disabled by issuing the ESTOP command. Asynchronous event processing is enabled by issuing the ASTART command, and disabled by issuing the ASTOP command.

It is possible to temporarily block asynchronous events from being processed using the ABLOCK command. This temporary block is released using the AUNBLOCK command. Events can be generated by running processes using the ETRIGGER command.

Asynchronous event processing and synchronous event processing cannot both be enabled at the same time for any event class.

6.3.4.1 Event classes

Events are divided into event classes, and those classes are further divided into event IDs. Each event class may be independently enabled, disabled, blocked, and unblocked (except that individual event

classes may not be disabled in the synchronous model).

The event classes aredescribed below.

6.3.4.1.1 COMM

The event class COMM contains events that are associated with devices. evid is always a devicexpr for this class of event. Not all devices necessarily generate events. What devices generate COMM events, and under what circumstances is determined by the implementation. It is to be understood that use of COMM events may not be portable.

6.3.4.1.2 HALT

The event class HALT contains events that are generated when a process terminates. evid is 1 for processes which halt by an explicit HALT command. Other values may be specified by the implementation to correspond to vendor-specific job termination utilities. It is to be understood that use of these other values may not be portable.

6.3.4.1.3 IPC

The event class IPC contains events that are generated by other processes using the ETRIGGER command. The evid values are restricted to valid processids. The evid value will always be the processid of the process that issued the ETRIGGER command.

6.3.4.1.4 INTERRUPT

The event class INTERRUPT contains events that are generated by the interruption of a running job in some implementation-specific manner (typically by implementation-specific keyboard commands or job control utilities). Different forms of interrupts may be possible in some implementations, and these may possibly be differentiated by evid values. The validity of evid values is determined by the implementor. It is to be understood that use of INTERRUPT events may not be portable

6.3.4.1.5 POWER

The event class POWER contains events that are generated when an imminent loss of power can be anticipated (typically because of a signal from the power source). Different types of warnings may be possible in some implementations, and these may possibly be differentiated by evid values. The evid values are determined by the implementor. It is to be understood that use of POWER events may not be portable.

6.3.4.1.6 TIMER

The event class TIMER contains events that are generated when a specified interval has elapsed after a timer was set (see ^$EVENT). evid values are names. The implementor may limit the number of concurrent timers available, either by a single process or by the entire M[UMPS] system, or both.

6.3.4.1.7 USER

The event class USER contains events that are generated by ETRIGGER commands in the current process. evid values are names.

6.3.4.1.8 Z[unspecified]

Z is the initial letter reserved for defining non-standard event classes. The requirement that Z be used permits the unused names to be reserved for future extensions to the standard without altering the execution of existing routines which observe the rules of the standard.

6.3.4.2 Event registration

Only those events which have been registered by creating a node in ^$JOB(processid ,"EVENT", evclass , evid) generate action. In those cases the value of the node is an labelref which specifies the event handler (see page 32).

6.3.4.3 Asynchronous event processing

Asynchronous processing of an event (described below) occurs immediately following the event unless the event is blocked.

Blocked events are saved on one of two per-process event queues (one each for synchronous and asynchronous event classes). Each queue is only guaranteed to hold one event, though they may hold more. Events occurring when the queue is full are lost. Queued events are processed in the order they occurred once they are unblocked. It is possible that blocked events will not execute in the order they occurred if the events are of different event classes, and the event classes are separately unblocked in an order different from the order of occurrence of the events. Disabling an event class via the ASTOP command or by killing the appropriate node(s) in ^$EVENT or ^$JOB removes all entries of that class from the event queue.

When a registered event is processed in the asynchronous model, the current value of $TEST, the current execution level, and the current execution location are saved in an extrinsic frame on the PROCESS-STACK. The process then increments the block count on all event classes, and implicitly executes the command

```
DO handler
```

where *handler* is the registered event handler. Note that neither $REFERENCE nor any other shared resource is stacked by this action. If the event handler changes the naked indicator, it may be advisable for it to first NEW $REFERENCE. When the process control returns from the handler, the process decrements the block count on all event classes. The value of $TEST and the execution level are restored, the process returns to the stacked execution location and the extrinsic frame is removed from the PROCESS-STACK.

6.3.4.4 Synchronous event processing

Synchronous event processing is enabled by the ESTART command, which leaves the process in a wait-for-event state. Events are processed sequentially in the order in which they occur. Each event is added to the per-process synchronous event queue. This queue is only guaranteed to hold one event, though it may hold more. Events occurring when the queue is full are lost. When the process is in the wait-for-event state and there is an event in the queue, the event is processed in the synchronous model.

When a registered event is processed in the synchronous model, the process implicitly executes the command

```
DO handler
```

where *handler* is the registered event handler. When process control returns from the handler, the process returns to the waiting-for-event state. If the handler executes an ESTOP command, the control implictly performs the number of QUIT commands necessary to return to the execution level of the most recently executed ESTART command, and then terminates that ESTART command.

When a process is initiated, no event processing is enabled, and no nodes in ^$JOB(processid ,"EVENT") are defined. When a process terminates, event processing is implicitly terminated and ^$JOB(processid , "EVENT") is implicitly killed. Any queued events (synchronous or asynchronous event queues) for that process are discarded.

6.4 Embedded programs

An embedded *xxx* M[UMPS] program is a program which consists of M[UMPS] text and text written to the specifications of the *xxx* programming language or standard. Although it is not a routine, an embedded M[UMPS] program conforms to the syntax of a M[UMPS] routinebody.

extsyntax ::= & extid (exttext)

exttext ::= | [graphic] ...
[graphic] ... [eol & ls [graphic] ...] ... |

extid ::= | SQL
Z[unspecified] |

In exttext each eol & ls sequence is either ignored or, if required by the other programming language or standard, replaced by one or more graphic characters. Exttext is then treated as if the graphic characters following the ls were part of the previous line (a continuation line).

The exact syntax of the remainder of exttext is defined by the external programming language or standard. In the case of extid being SQL this standard is ANSI X3.135 (see also Annex D). extids differing only in the use of corresponding upper and lower case letters are equivalent. extids not beginning with the letter Z are reserved for future extensions to the language.

Note: An embedded program implies that one or more M[UMPS] routines may be created by some compilation process, replacing any external syntax with appropriate M[UMPS] command lines, function calls et cetera. An embedded program or embedded program pre-processor does not, therefore, need to adhere to the portability requirements of Section 2 although the equivalent M[UMPS] routines and M[UMPS] implementation should.

7 . Expression expr

The expression, expr, is the syntactic element which denotes the execution of a value-producing calculation. Expressions are made up of expression atoms separated by binary, string, arithmetic, or truth-valued operators.

expr ::= expratom [exprtail] ...

7.1 Expression atom expratom

The expression atom, expratom, is the basic value-denoting object of which expressions are built.

expratom ::= | glvn
expritem
owservice |

7.1.1 Values and Variables

7.1.1.1 Values

All expressions and defined variables have values. These values, whether intermediate or final, may be thought of and operated upon as one of two data types, MVAL or OREF.

7.1.1.1.1 Values of data type MVAL

mval ::= any value with data type MVAL

Values of data type MVAL are values that may be operated upon as strings.

7.1.1.1.2 Values of data type OREF

oref ::= any value with data type OREF

Values of data type OREF are values that have no canonic representation. Instead, each oref uniquely identifies a specific *object*, in an implementation-specific manner.

When a value of data type OREF is assigned to an lvn, this value assignment is the only action that will result. In particular, no copy is made of the *object* that is identified by this oref.

All copies of a value of data type OREF are completely equivalent for accessing the *object*.

After an lvn has been assigned a value of data type OREF, any new value may be assigned to that lvn. Such new values may be of data type MVAL as well as OREF. Such assignments never have any impact on the object that is identified by the original oref.

Values of data type OREF have no literal representation. Except in certain special situations, values of data type OREF are coerced into values of data type MVAL according to the following rules, based on the value of the default property, if any, of the object identified by the oref:

a. Let *ref* be the value of data type OREF that is being coerced.

b. Let *obj* be the *object* identified by *ref.*

c. If *obj* has no default *property*, an error condition will occur with ecode = "M107" (No Default Value).

d. If the value of the default *property* of *obj* has the data type OREF, then replace *ref* by the value of this default *property* and go back to step b.

e. Otherwise, the value of the default *property* of *obj* must be of data type MVAL, and this value will be returned as the coerced string value of the original *object*.

The special situations under which values of data type OREF are not coerced into values of data type MVAL are:

1. The final values of the exprs that are operands of == operators.

2. The values of parameters that are passed in actuallists or namedactuallists that are not part of externrefs or JOB command arguments.

3. The final values of the exprs on the right-hand side of the = in arguments of ASSIGN commands.

4. Values that are returned as function-values through QUIT commands.

5. Values that are used as the object portion of an owservice.

7.1.1.2 Variables

The M[UMPS] standard uses the terms *local variables* and *global variables* somewhat differently from their connotation in certain other computer languages. This subclause provides a definition of these terms as used in the M[UMPS] environment.

A M[UMPS] routine, or set of routines, runs in the context of an operating system process. During its execution, the routine will create and modify variables that are restricted to its process. These process-specific variables may be stored in primary memory or on secondary peripheral devices such as disks. It can also access (or create) variables that can be shared with other processes. These shared variables will normally be stored on secondary peripheral devices such as disks. At the termination of the process, the process-specific variables cease to exist. The variables created for long term (shared) use remain on auxiliary storage devices where they may be accessed by subsequent processes.

M[UMPS] uses the term *local variable* to denote variables that are created for use during a single process activation. These variables are not available to other processes. However, they are generally available to all routines executed within the process's lifetime. M[UMPS] does include certain constructs, the NEW command and parameter passing, which limit the availability of certain variables to specific routines or parts of routines.

A *global variable* is one that is created by a process, but is permanent and shared. As soon as a process creates, modifies or deletes a global variable outside of a TRANSACTION, other processes accessing that global variable outside of a TRANSACTION receive its modified form. (See 6.3.1 Transaction processing for a definition of TRANSACTION and information on how TRANSACTIONs affect global variable modifications.) Global variables do not disappear when a process terminates. Like local variables, global variables are available to all routines executed within a process.

M[UMPS] has no explicit declaration or definition statements. Local and global variables, both non-subscripted and subscripted, are automatically created as data is stored into them, and their data contents can be referred to once information has been stored. Since most operations in the language create values of only one data type - string - there is no need for type declarations or explicit data type conversions. Array structures can be multidimensional with data simultaneously stored at all levels including the variable name level. Subscripts can be positive, negative, or zero; they can be integer or non-integer numbers as well as non-numeric strings (other than empty strings).

7.1.2 Variable name glvn

The metalanguage element glvn is defined so as to be satisfied by the syntax of gvn, lvn, or ssvn.

```
glvn ::= | lvn
         | gvn
         | ssvn |
```

7.1.2.1 Local variable name lvn

```
lvn ::= | rlvn
        | @ expratom V lvn |

rlvn ::= | name [ ( L expr ) ]
         | @ lnamind @ ( L expr ) |

lnamind ::= rexpratom V lvn

rexpratom ::= | rlvn
              | rgvn
              | rssvn
              | expritem |
```

See 7.1.2.4 for the definition of rgvn. See 7.1.4 for the definition of expritem.

A local variable name is either unsubscripted or subscripted; if it is subscripted, any number of subscripts separated by commas is permitted. Except where otherwise specified, a subscript may not equal the empty string. An unsubscripted occurrence of lvn may carry a different value from any subscripted occurrence of lvn.

When lnamind is present it is always a component of an rlvn. If the value of the rlvn is a subscripted form of lvn, then some of its subscripts may have originated in the lnamind. In this case, the subscripts contributed by the lnamind appear as the first subscripts in the value of the resulting rlvn, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rlvn.

7.1.2.2 Local variable handling

In general, the operation of the local variable symbol table can be viewed as follows. Prior to the initial setting of information into a variable, the data value of that variable is said to be undefined. Data is stored into a variable with commands such as ASSIGN, FOR, JOB, MERGE, READ, SET, TCOMMIT, TRESTART, and TROLLBACK. Subsequent references to that variable return the data value that was most recently stored. When a variable is killed, as with the KILL command, that variable and all of its array descendants (if any) are deleted, and their data values become undefined.

No explicit syntax is needed for a routine or subroutine to have access to the local variables of its caller. Except when the NEW command or parameter passing is being used, a subroutine or called routine (the callee) has the same set of variable values as its caller and, upon completion of the called routine or subroutine, the caller resumes execution with the same set of variable values as the callee had at its completion.

The NEW command provides scoping of local variables. It causes the current values of a specified set of variables to be saved. The variables are then set to undefined data values. Upon returning to the caller of

the current routine or subroutine, the saved values, including any undefined states, are restored to those variables. Parameter passing, including the DO command, extrinsic functions, and extrinsic variables, allows parameters to be passed into a subroutine or routine without the callee being concerned with the variable names used by the caller for the data being passed or returned.

The formal association of local variables with their values can best be described by a conceptual model. This model is NOT meant to imply an implementation technique for a M[UMPS] implementation.

The value of a variable may be described by a relationship between two structures: the NAME-TABLE and the VALUE-TABLE. (In reality, at least two such table sets are required, one pair per executing process for process-specific local variables and one pair for system-wide global variables.) Since the value association process is the same for both types of variables, and since issues of scoping due to parameter passing or nested environments apply only to local variables, the discussion that follows will address only local variable value association. It should be noted, however, that while the overall structures of the table sets are the same, there are two major differences in the way the sets are used. First, the global variable tables are shared. This means that any operations on the global variable tables, e.g., SET or KILL, by one process, affect the tables for all processes. Second, since scoping issues of parameter passing and the NEW command are not applicable to global variables, there is always a one-to-one relationship between entries in the global NAME-TABLE (variable names) and entries in the global VALUE-TABLE (values).

The NAME-TABLE consists of a set of entries, each of which contains a name and a pointer. This pointer represents a correspondence between that name and exactly one DATA-CELL from the VALUE-TABLE. The VALUE-TABLE consists of a set of DATA-CELLs, each of which contains zero or more tuples of varying degrees. The degree of a tuple is the number (possibly 0) of elements or subscripts in the tuple list. Each tuple present in the DATA-CELL has an associated data value.

The NAME-TABLE entries contain every non-subscripted variable or array name (name) known, or accessible, by the process in the current environment. The VALUE-TABLE DATA-CELLs contain the set of tuples that represent all variables currently having data-values for the process. Every name (entry) in the NAME-TABLE refers (points) to exactly one DATA-CELL, and every entry contains a unique name. Several NAME-TABLE entries (names) can refer to the same DATA-CELL, however, and thus there is a many-to-one relationship between (all) NAME-TABLE entries and DATA-CELLs. A name is said to be *bound* to its corresponding DATA-CELL through the pointer in the NAME-TABLE entry. Thus the pointer is used to represent the correspondence and the phrase *change the pointer* is the equivalent to saying *change the correspondence so that a name now corresponds to a possible different DATA-CELL (value)*. NAME-TABLE entries are also placed in the PROCESS-STACK (see 7.1.2.3 Process-Stack).

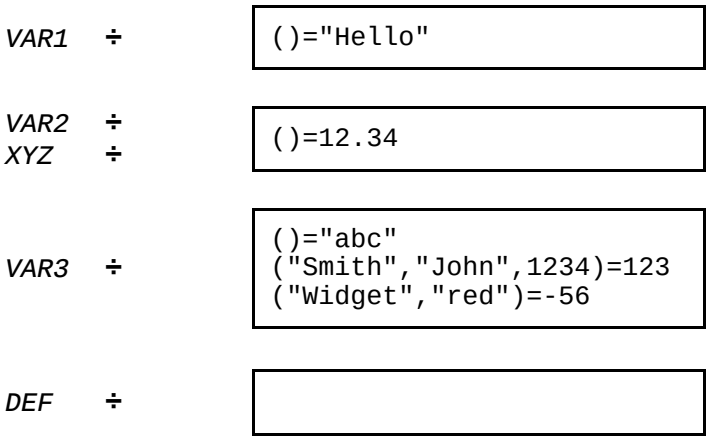
The value of an unsubscripted lvn corresponds to the tuple of degree 0 found in the DATA-CELL that is bound to the NAME-TABLE entry containing the name of the lvn. The value of a subscripted lvn (array node) of degree n also corresponds to a tuple in the DATA-CELL that is bound to the NAME-TABLE entry containing the name of the lvn. The specific tuple in that DATA-CELL is the tuple of degree n such that each subscript of the lvn has the same value as the corresponding element of the tuple. If the designated tuple doesn't exist in the DATA-CELL then the corresponding lvn is said to be *undefined*.

In the following figure, the variables and array nodes have the designated data values.

VAR1 = "Hello"
VAR2 = 12.34
VAR3 = "abc"
VAR3("Smith","John",1234)=123
VAR3("Widget","red") = -56

Also, the variable *DEF* existed at one time but no longer has any data or array value, and the variable *XYZ* has been bound through parameter passing to the same data and array information as the variable *VAR2*.

NAME-TABLE VALUE-TABLE DATA-CELLS



The initial state of a process prior to execution of any M[UMPS] code consists of an empty NAME-TABLE and VALUE-TABLE. When information is to be stored (set, given, or assigned) into a variable (lvn):

a. If the name of the lvn does not already appear in an entry in the NAME-TABLE, an entry is added to the NAME-TABLE which contains the name and a pointer to a new (empty) DATA-CELL. The corresponding DATA-CELL is added to the VALUE-TABLE without any initial tuples.

b. Otherwise, the pointer in the NAME-TABLE entry which contained the name of the lvn is extracted. The operations in steps c and d refer to tuples in that DATA-CELL referred to by this pointer.

c. If the lvn is unsubscripted, then the tuple of degree 0 in the DATA-CELL has its data value replaced by the new data value. If that tuple did not already exist, it is created with the new data value.

d. If the lvn is subscripted, then the tuple of subscripts in the DATA-CELL (i.e., the tuple created by dropping the name of the lvn; the degree of the tuple equals the number of subscripts) has its data value replaced by the new data value. If that tuple did not already exist, it is created with the new data value.

When information is to be retrieved, if the name of the lvn is not found in the NAME-TABLE, or if its corresponding DATA-CELL tuple does not exist, then the data value is said to be undefined. Otherwise, the data value exists and is retrieved. A data value of the empty string (a string of zero length) is not the same as an undefined data value.

When a variable is deleted (killed):

a. If the name of the lvn is not found in the NAME-TABLE, no further action is taken.

b. If the lvn is unsubscripted, all of the tuples in the corresponding DATA-CELL are deleted.

c. If the lvn is subscripted, let *N* be the degree of the subscript tuple formed by removing the name from the lvn. All tuples that satisfy the following two conditions are deleted from the corresponding DATA-CELL:

    1. The degree of the tuple must be greater than or equal to *N*, and

    2. The first *N* descriptors of the tuple must equal the corresponding subscripts of the lvn.

In this formal language model, even if all of the tuples in a DATA-CELL are deleted, neither the DATA-CELL nor the corresponding names in the NAME-TABLE are ever deleted. Their continued existence is frequently required as a result of parameter passing and the NEW command.

7.1.2.3 Process-Stack

The PROCESS-STACK is a virtual last-in-first-out (LIFO) list (a simple push-down stack) used to describe the behavior of M[UMPS]. It is used as an aid in describing how M[UMPS] appears to work and does not imply that an implementation is required to use such a stack to achieve the specified behavior. Three types of items, or frames, will be placed on the PROCESS-STACK, DO frames (including XECUTEs), extrinsic frames (including exfunc and exvar and asynchronous events) and error frames (for errors that occur during error processing):

a. DO frames contain the execution level and the execution location of the doargument or xargument. In the case of the argumentless DO, the execution level, the execution location of the DO command and a saved value of $TEST are saved. The execution location of a process is a descriptor of the location of the command and possible argument currently being executed. This descriptor includes, at minimum, the routinename and the character position following the current command or argument.

b. Extrinsic frames contain saved values of $TEST, the execution level, and the execution location.

c. Error frames contain information about error conditions during error processing (see 6.3.2 Error processing).

The term CONTEXT-STRUCTURE is used to refer to a set of information related to the maintenance of the process context.

7.1.2.4 Global variable name gvn

```
gvn ::= | rgvn
        | @ expratom V gvn |

rgvn ::= | ^( L expr )
         | ^ [ VB environment VB ] name [ ( L expr ) ]
         | @ gnamind @ ( L expr ) |

gnamind ::= rexpratom V gvn

environment ::= expr
```

The prefix ^ uniquely denotes a global variable name. A global variable name is either unsubscripted or subscripted; if it is subscripted, any number of subscripts separated by commas is permitted. Except where otherwise specified, a subscript may not equal the empty string. An abbreviated form of subscripted gvn is permitted, called the *naked reference*, in which the prefix is present but the environment, name and an initial (possibly empty) sequence of subscripts is absent but implied by the value of the *naked indicator*. An unsubscripted occurrence of gvn may carry a different value from any subscripted occurrence of gvn.

When environment is present it identifies a specific set of all possible names.

When gnamind is present it is always a component of an rgvn. If the value of the rgvn is a subscripted form of gvn, then some of its subscripts may have originated in the gnamind. In this case, the subscripts contributed by the gnamind appear as the first subscripts in the value of the resulting rgvn, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rgvn.

Every executed occurrence of gvn affects the naked indicator as follows. If, for any positive integer *m*, the gvn has the non-naked form

$N(v_1 , v_2 , ... , v_m)$

then the *m*-tuple $N, v_1 , v_2 , ... , v_{m!1}$, is placed into the naked indicator when the gvn reference is made. A subsequent naked reference of the form

$^(s_1 , s_2 , ... , s_i)$ (*i* positive)

results in a global reference of the form

$N(v_1 , v_2 , ... , v_{m!1} , s_1 , s_2 , ... , s_i)$

after which the $m+i!1$-tuple $N , v_1 , v_2 , ... , s_{i!1}$ is placed into the naked indicator. Prior to the first executed occurrence of a non-naked form of gvn, the value of the naked indicator is undefined. A non-naked reference without subscripts or a ROLLBACK, or a change of the default global variable environment leaves the naked indicator undefined. When a gvn is encountered in the form of a naked reference and the naked indicator is undefined, an error condition occurs with ecode = "M1".

The effect on the naked indicator described above occurs regardless of the context in which gvn is found; in particular, an assignment of a value to a global variable with the command SET gvn = expr does not affect the value of the naked indicator until after the right-side expr has been evaluated. The effect on the naked indicator of any gvn within the right-side expr will precede the effect on the naked indicator of the left-side gvn.

7.1.3 Structured system variable ssvn

ssvn ::= | rssvn
@ expratom V ssvn |

rssvn ::= | ^$ [VB environment VB] ssvname [(L expr)]
@ ssvnamind @ (L expr) |

ssvnamind ::= rexpratom V ssvn

The prefix ^$ uniquely denotes a structured system variable name. The parenthesized list of exprs following the ssvname are called subscripts; a ssvn may be either subscripted or unsubscripted; if it is subscripted, any number of subscripts separated by commas is permitted. The allowed values and/or interpretation of each subscript is defined for each individual ssvname; except where otherwise specified, a subscript may not equal the empty string. Structured system variable names (ssvnames) differing only in the use of corresponding upper and lower case letters are equivalent.

When ssvnamind is present it is always the component of a rssvn. If the value of the rssvn is a subscripted form of ssvn, then some of its subscripts may have originated in the ssvnamind. In this case, the subscripts contributed by the ssvnamind appear as the first subscripts in the value of the resulting rssvn, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rssvn.

Values may not be assigned to ssvns and ssvns may not be KILLed unless the semantics of these operations are explicitly defined. The environment form of the ssvn syntax may only refer to the default environment unless the ssvn is explicitly defined to permit the use of environments other than the default. A reference to such an ssvn which refers to an environment that is not explicitly permitted is erroneous and causes an error condition with ecode = "M59". Other references to ssvns using the environment syntax however, due to technical reasons or security concerns, may be restricted by implementors to a restricted set of possible environments. An attempt to violate this restriction causes an error condition with an implementor-specified ecode beginning with "Z".

The meaning of the individual subscripts of a ssvn is explicitly defined for each ssvn. The standard contains the following ssvnames:

ssvname ::= C[HARACTER] | D[EVICE] | E[VENT] | G[LOBAL] | J[OB] | LI[BRARY] | L[OCK] | R[OUTINE] | S[YSTEM] | Y[unspecified] | Z[unspecified]

Unused structured system variable names beginning with an initial letter other than Z are reserved for future extensions to the standard.

7.1.3.1 ^$CHARACTER

^$C[HARACTER] (charsetexpr)

charsetexpr ::= expr V charset

^$CHARACTER provides information regarding the available Character Set Profiles on a system, such as collation order and pattern code definitions.

If and only if a Character Set Profile identified by charset exists, ^$CHARACTER(charset) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard.

Data manipulation and the execution of commands within a process are performed in the context of the process charset. (See 7.1.3.4 ^$JOB)

7.1.3.1.1 Input-Transformation

^$CHARACTER($charsetexpr_1$, $expr_1$ V "INPUT" , $charsetexpr_2$) = $expr_2$ V algoref

algoref ::= emptystring | $$ labelref | $ externref | $ functionname

emptystring ::= a string of zero length

This node specifies the input-transformation algorithm which is performed on a string in the process Character Set Profile $charsetexpr_1$ when it is retrieved from a global variable or routine which uses $charsetexpr_2$ or transmitted from a device using $charsetexpr_2$. Let *transform* be the value of $expr_2$. *transform* specifies the algorithm by which this translation is accomplished, if no input-transformation algorithm is defined, an empty-string value is used. The conversion of the string *old* to the string *new* using the input-transformation algorithm *transform* may be evaluated by executing:

```
("S new="_transform_"(old)")
```

7.1.3.1.2 Output-Transformation

^$CHARACTER(charsetexpr$_1$, expr$_1$ V "OUTPUT" , charsetexpr$_2$) = expr$_2$ V algoref

This node specifies the output-transformation algorithm which is performed on a string in the process Character Set Profile charsetexpr$_1$ when it is stored in a global variable or routine which uses charsetexpr$_2$ or transmitted to a device using charsetexpr$_2$. Let *transform* be the value of expr$_2$. *transform* specifies the algorithm by which this translation is accomplished, if no output-transformation algorithm is defined, an empty-string value is used. The conversion of the string *old* to the string *new* using the output-transformation algorithm *transform* may be evaluated by executing:

```
("S new="_transform_"(old)")
```

7.1.3.1.3 Valid name characters

^$CHARACTER(charsetexpr , expr$_1$ V "IDENT") = expr$_2$ V algoref

This node specifies the identification algorithm used to determine which characters in a charset are valid for use in names (i.e. is a character in the set ident).

The ident truth-value *truth*, of a character *char* using an identification algorithm *ident*, may be evaluated by executing the expression:

```
("S truth="_ident_"($ASCII(char))")
```

When *truth* is "true", *char* is an ident; when *truth* is "false", *char* is not an ident.

Note that digits are implicitly allowed in names and that for $ASCII(*char*) values less than 128, 65-90 and 97-122 are required to be "true" and all other values less than 128 are required to be "false". If the identification algorithm node is undefined, or is the empty string, then it will return "false" for all $ASCII(*char*) greater than 127; values less than 128 will be returned as indicated.

7.1.3.1.4 patcode definition

^$CHARACTER(charsetexpr , expr$_1$ V "PATCODE" , expr$_2$ V patcode) = expr$_3$ V algoref

This node identifies the pattern testing algorithm that determines which characters of charsetexpr match the specified patcode; if this node is not defined, or is the empty string, then no characters in the charsetexpr will match that patcode. Let *pattest* be the value of expr$_3$. The patcode truth-value *truth* of a character *char* using a non-empty-string pattern testing algorithm *pattest* may be evaluated by executing the expression:

```
("S truth="_pattest_"($ASCII(char))")
```

When *truth* is "true", *char* belongs to the specified patcode; when *truth* is "false", *char* does not belong to that patcode.

7.1.3.1.5 Collation Algorithm

^$CHARACTER(charsetexpr , expr$_1$ V "COLLATE") = expr$_2$ V algoref

This node identifies the collation algorithm for the specified Character Set Profile (charset).
See 7.1.5.13 ($ORDER) for a description of the model that is assumed for this algorithm.

Editor's note:
Suggest to include an example similar to the ones in the previous clauses:
The internal collating value for a character set may be evaluated by executing the expression:

```
("S internal="_collate_"(external)")
```

7.1.3.1.6 Case Conversion

^$CHARACTER (charsetexpr , $expr_1$ V “LOWER”) = $expr_2$ V algoref

This optional node identifies the algorithm for the conversion of character strings whereby upper-case characters are converted to lower-case ones.

^$CHARACTER (charsetexpr , $expr_1$ V “UPPER”) = $expr_2$ V algoref

This optional node identifies the algorithm for the conversion of character strings whereby lower-case characters are converted to upper-case ones.

Editor’s note:
Suggest to include an example similar to the ones in the previous clauses:
The result of a conversion to lower or upper case for a character set may be evaluated by executing the expression:

```
(“S result=”_lower_“(string)”)
(“S result=”_upper_“(string)”)
```

7.1.3.2 ^$DEVICE

^$D[EVICE] (devicexpr)

devicexpr ::= expr V device

device ::= devicespecifier; an implementation-specific device identifier

^$DEVICE provides information about the existence, operational characteristics and availability of devices.

Note: The holding of information about a device when it is not open may be transitory. There are also likely to be more devices in a system which could be opened by a M[UMPS] process than will have information stored in ^$DEVICE.

Device characteristic information for a device is stored beneath the ^$DEVICE(devicexpr) node:

7.1.3.2.1 Character set for device

^$DEVICE(devicexpr , expr V "CHARACTER") = charsetexpr

This node identifies the current Character Set Profile of the specified device. The Character Set Profile is assigned to the device in an implementation-specific manner.

7.1.3.2.2 Device attributes

^$DEVICE (devicexpr , expr V deviceattribute)

This contains the primary value or values associated with this deviceattribute. Additional values may be stored in descendants of this node.

When a device is opened then values for the deviceattributes are created in ^$DEVICE. These may be retained after the device is closed. The range of deviceattribute names and the format of the values is defined by the mnemonicspace in use for the device.

7.1.3.2.3 Format functions

^$DEVICE (devicexpr , $expr_1$ V "MNEMONICSPACE") = $expr_2$ V mnemonicspace

This node identifies the mnemonicspace currently in effect for the device. If there is no mnemonicspace in effect then this node has the value of the empty string.

^$DEVICE (devicexpr , $expr_1$ V "MNEMONICSPEC" , $expr_2$ V mnemonicspace) = emptystring

This node identifies a mnemonicspace that has been associated with the device through the OPEN and USE commands. All non-empty string values are reserved for future extensions to the standard.

When the mnemonicspace in use for the device defines an output timeout as described in 8.3.1, it shall also define the following two members of ^$DEVICE:

a. the value of ^$DEVICE (devicexpr , expr V “OUTTIMEOUT”) shall equal the value of the most recently executed OUTTIMEOUT deviceparam for the device. It shall equal 0 when no OUTTIMEOUT deviceparam has executed for the device.

b. the value of ^$DEVICE (devicexpr , expr V “OUTSTALLED”) shall indicate the output timeout status of the device. It shall assume the value 0 when the execution of any output-producing argument of a READ or WRITE command begins, and it shall assume the value 1 when that argument times out.

7.1.3.2.4 Sockets

The following nodes are defined in ^$DEVICE for the SOCKET mnemonicspace:

^$DEVICE(devicexpr , expr V “SOCKET”) = intexpr

Each device has a collection of sockets associated with it. Each new socket is identified by a socket identifier which is assigned an index number in the collection of sockets. This node of ^$DEVICE defines the index number of the current socket.

^$DEVICE(devicexpr , expr V “SOCKET”, intexpr , $expr_2$ V “DELIMITER”) = $intexpr_2$

This provides the number of I/O delimiters, as defined using the DELIMITER deviceattribute, in effect for the device/socket. (See Appendix H, 3.1.3)

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, $intexpr_1$, $expr_2$ V “DELIMITER”, $intexpr_2$) = expr

This provides the *n*-th I/O delimiter string. (See Appendix H, 3.1.3)

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, intexpr , $expr_2$ V “IOERROR”) = $expr_3$

I/O error trapping mode. (See Appendix H, 3.1.4)

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, intexpr , $expr_2$ V “LOCALADDRESS”) = $expr_3$

This provides the local network node address of the connection

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, intexpr , $expr_2$ V “PROTOCOL”) = $expr_3$

This provides the network protocol used for the connection

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, intexpr , $expr_2$ V “REMOTEADDRESS”) = $expr_3$

This provides the remote network node address of the connection

^$DEVICE(devicexpr , $expr_1$ V “SOCKET”, intexpr , $expr_2$ V “SOCKETHANDLE”) = $expr_3$

The value of this node is an implementation-specific string that provides the socket identifier of the indicated socket.

7.1.3.3 ^$EVENT

^$E[VENT] (eventexpr)

eventexpr ::= expr V | “EVENTDEF” einfoattribute |

Note that einfoattribute is defined in X11.6, the MWAPI standard, along with its semantics. Nodes under ^$EVENT(einfoattribute) are used to identify specific behavior of MWAPI events.

Nodes under ^$EVENT("EVENTDEF") are used to identify specific behavior of the named events.

7.1.3.3.1 Timer Events:

^$EVENT($expr_1$ V "EVENTDEF" , $expr_2$ V "TIMER")

^$EVENT("EVENTDEF", "TIMER") provides information about the characteristics of the TIMER class of events. Let *timerid* be the value of a valid evid for a TIMER event. All of the nodes below must be set to establish the timer. If any of these nodes are killed, no timer event occurs.

Timer Event Interval:

^$EVENT($expr_1$ V "EVENTDEF" , $expr_2$ V "TIMER" , *timerid* , $expr_3$ V "INTERVAL") = intexpr

This node identifies (if positive) the running time remaining before the timer event (in seconds). This value counts down continuously at a rate of 1 per second. While the value of this node is greater than 0, the value of the corresponding node ^$EVENT("EVENTDEF", "TIMER", *timerid* , "ACTIVE") (see below) evaluates as a tvexpr to 1.

Timer Event Interval Auto-Reset:

^$EVENT($expr_1$ V "EVENTDEF" , $expr_2$ V "TIMER" , *timerid* , $expr_3$ V "AUTO") = intexpr

This node is the value set into ^$EVENT("EVENTDEF", "TIMER", *timerid* , "INTERVAL") when it is decremented from a positive value to a non-positive value.

Timer Event State:

^$EVENT($expr_1$ V "EVENTDEF" , $expr_2$ V "TIMER" , *timerid* , $expr_3$ V "ACTIVE") = tvexpr

This node identifies the state of the timer. If the node evaluates as a tvexpr to 1, the timer is active (running). If the node evaluates as a tvexpr to 0, the timer is inactive.

7.1.3.4 ^$GLOBAL

^$G[LOBAL] (gvnexpr)

gvnexpr ::= expr V | ^ name |

^$GLOBAL provides information about the existence and characteristics of global variables.

If and only if a global variable identified by gvnexpr exists, ^$GLOBAL(gvnexpr) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard. Global variable characteristic information is stored beneath the ^$GLOBAL(gvnexpr) node:

^$GLOBAL(gvnexpr , expr V "CHARACTER") = charsetexpr

This node identifies the Character Set Profile of the specified global variable. When the first node in a global variable is created, and the node ^$GLOBAL(gvnexpr , "CHARACTER") has a $DATA value of zero, the value assigned is that of ^$JOB($JOB, "CHARACTER"). The result of killing a gvn does not alter the characteristics stored in ^$GLOBAL for that gvn.

7.1.3.4.1 Collation Algorithm

^$GLOBAL(gvnexpr , $expr_1$ V "COLLATE") = $expr_2$ V algoref

This node identifies the collation algorithm to be used when collation is required for a reference to this global variable. Let *collate* be the value of $expr_2$. The collation value *order* for a subscript-string *subscript*, and a collation algorithm *collate* may be determined by executing the expression:

```
("S order="_collate_"(subscript )")
```

In all cases a collation algorithm must return a distinct *order* for each distinct *subscript*.

When the first node of a global variable *^global* is created, and the collation algorithm node ^$GLOBAL(*"^global"* , "COLLATE") has a $DATA value of zero, then the value of the current process' Character Set Profile collation algorithm ($GET(^$CHARACTER(^$JOB($JOB, "CHARACTER"), "COLLATE"))) is assigned as the collation algorithm of the global variable (^$GLOBAL("*^global "* , "COLLATE")).

7.1.3.5 ^$JOB

^$J[OB] (processid)

processid ::= expr V jobnumber

^$JOB provides information about the existence and characteristics of processes in a system.

If and only if a process identified by processid exists, ^$JOB(processid) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard. Process characteristics are stored beneath the ^$JOB(processid) node.

7.1.3.5.1 Characteristic: Character Set Profile

^$JOB(processid , expr V "CHARACTER") = charsetexpr

This node identifies the active Character Set Profile in use by the process indicated by processid. Unless otherwise modified via the processparameters of the JOB command, when a process is created ^$JOB($JOB, "CHARACTER") is set to the charset of the process that created it.

7.1.3.5.2 Characteristic: Available Function Libraries

^$JOB(processid , $expr_1$ V "LIBRARY" , $expr_2$) = libraryexpr

This node identifies a library currently available to the process. The order in which the librarys are searched to locate a specific libraryelement is defined by the collating order of the values of $expr_2$ for the

specified librarys.

Editor's note:
The standard states that SETting and KILLing ssvns is prohibited, unless the semantics of such operations are explicitly defined. This is one location where the effects of SETs and KILLs should be included.

7.1.3.5.3 Characteristic: Devices

^$JOB(processid , expr V "$PRINCIPAL") = devicexpr (principal device)

^$JOB(processid , expr V "$IO") = devicexpr (current device)

^$JOB(processid , expr V "OPEN" , devicexpr) = (for each OPENed device)

These nodes specify the device information associated with process processid. The value of node "$PRINCIPAL" is equal to the value of $PRINCIPAL for that process. The value of node "$IO" is equal to the value of $IO (current device being used) for that process. The devicexpr nodes beneath "OPEN" are the device identifiers for all the devices which are currently OPENed for that process.

7.1.3.5.4 Characteristic: User and User Group

^$JOB(processid , $expr_1$ V "USER") = $expr_2$

^$JOB(processid , $expr_1$ V "GROUP") = $expr_2$

If and only if the process identified by processid is associated by the implementation with a user for security purposes, the value of the node "USER" is an implementation-specific unambiguous identifier of the user owning the process.

If and only if the process identified by processid is associated by the implementation with a group of users for security purposes, the value of the node "GROUP" is an implementation-specific unambiguous identifier of a user group to which the user owning the process belongs.

These are write-once ssvns. At the time of process initiation as the result of execution of a JOB command, the ssvn values associated with the initiating process are copied to the ssvns associated with the new process's processid unless overridden, in an implementation-specific manner, by the processparameters on the JOB command's jobargument. If a node has a $DATA value of 0 or 10, the process may create the node and assign an unconstrained value to it. When a node has a $DATA value of 1 or 11, a value may not be assigned nor may the node be KILLed; when a process attempts to do so an error occurs with ecode = "M96". At the termination of the process identified by processid, these ssvns become undefined.

7.1.3.5.5 Characteristic: Events

Registered Events:

^$JOB (processid , $expr_1$ V "EVENT" , $expr_2$ V evclass , evid) = $expr_3$ V labelref

This node identifies the events enabled for event processing under either the synchronous or asynchronous event processing models, and specifies the event handler that is invoked to process the event. SETting this node enables the specified events for event processing. KILLing this node disables the specified events for event processing, and removes all child nodes, even if KVALUE is used.

Implementations are expected to support all of the specified evclass and evid values with the understanding that some events may never occur on a given implementation. If an evclass or evid not defined in the standard is used, an error condition occurs with ecode = "M38". Attempting to set this node

when evid cannot be registered due to resource availability will generate an error condition with ecode = "M110".

Event Processing Modes:

^$JOB (processid , $expr_1$ V "EVENT" , $expr_2$ V evclass , evid , $expr_3$ V "MODE") = | “DISABLED” “SYNCHRONOUS” “ASYNCHRONOUS” |

This node identifies the processing mode for the specified event by the specified process. If the specified event class is currently enabled for asynchronous event processing by this process (see page 95, ASTART), the value of the node will be "ASYNCHRONOUS". If the specified event class is currently enabled for synchronous event processing by this process (see page 98, ESTART), the value of the node will be "SYNCHRONOUS". If the specified event class is not enabled for either form of processing by this process, the value of the node will be "DISABLED".

Event Block Counts:

^$JOB (processid , $expr_1$ V “EVENT” , $expr_2$ V evclass , evid , $expr_3$ V “BLOCKS”) = intlit

This node gives the count of blocks (see page 94, ABLOCK, and page 96, AUNBLOCK) on the specified event for the specified process. It only exists if the event class is enabled in either synchronous or asynchronous event processing modes. If the value of the node is zero, the events are not blocked. If the value is greater than zero, the events are blocked.

7.1.3.5.6 Characteristic: default environments

The following ssvns, specifying default environments, are defined. This clause pertains to the following five ssvns:

^$JOB(processid , "DEVICE")	default device environment
^$JOB(processid , "GLOBAL")	default global variable environment
^$JOB(processid , "JOB")	default JOB environment
^$JOB(processid , "LOCK")	default lock environment
^$JOB(processid , "ROUTINE")	default routine environment

A process may always obtain and assign a value to these nodes, where processid = $JOB. However, for technical reasons or security concerns, implementations may restrict access to these nodes for processids other than the current processid. An attempt to violate this restriction causes an error condition with an implementor-specified ecode beginning with "Z".

When a process starts, the values of these ssvns are in general defined by the implementation. However, a process initiated by a JOB command inherits the default environments of the initiating process, unless explicitly specified otherwise in the jobargument.

Explicit qualification of a labelref, routineref, gvn, nref, or devn with an environment overrides the default environment for that one reference.

Assigning a non-existent environment to one of these ssvns is not in itself erroneous. However, an attempt to refer to a routine, global variable, lock, or device in a non-existent environment causes an error condition with an ecode = "M26".

7.1.3.5.7 Characteristic: Local variables

^$JOB (processid , expr V “VAR” , lvnexpr)

lvnexpr ::= name

For all local variables in the context of the specified processid, a node exists in this subtree of ^$JOB. The value of $DATA for any of these nodes is determined by the value of $DATA for the local variables in question. Likewise, the value of these nodes, if any, is the same as the value of the local variables in question (including undefined). Only local variables for which $DATA returns a non-zero value are represented by these nodes. If a lvn is of the form *Name*(S_1,S_2,...,S_n) for a process *Job*, then a reference to that lvn behaves identical to a reference to ^$JOB(*Job*,”VAR”,”*Name*”,S_1,S_2,...,S_n).

Coordination issues may arise if these nodes are examined by another process (if permitted by the implementation). A specific reference may be atomic, but multiple references are not – the local variable being examined may be NEWed or KILLed between two references to these nodes.

Note that for technical reasons or for security concerns, implementations may restrict access to nodes in ^$JOB for processids other than the current processid. An attempt to violate such a restriction will cause an error condition with an implementor-specified ecode, beginning with “Z”.

7.1.3.5.8 Characteristic: Localized Formatting

If an implementation provides the function $%FORMAT^STRING (see page 75), the following nodes may be defined.

^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”CS”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”DC”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”EC”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”FS”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”FM”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”SL”) = $expr_3$
^$JOB (processid , $expr_1$ V ”FORMAT” , $expr_2$ V ”SR”) = $expr_3$

The values of these nodes will provide process-wide defaults for the various localized formatting parameters. Possible values for these nodes and their meanings are described on page 75.

7.1.3.6 ^$LIBRARY

^$LI[BRARY] (libraryexpr)

libraryexpr ::= expr V library

^$LIBRARY provides information about the availability of libraries and library elements in a system.

If and only if a library identified by libraryexpr exists, ^$LIBRARY(libraryexpr) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard. Library information is stored beneath the ^$LIBRARY(library) node:

^$LIBRARY (libraryexpr , expr V "ELEMENT" , libraryelementexpr)

libraryelementexpr ::= expr V libraryelement

Let *lib* be the value of libraryexpr and let *e* be the value of libraryelementexpr. If and only if a library *lib* and a libraryelement *e* exist, ^$LIBRARY(*lib* , "LIBRARY", *e*) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard.

7.1.3.7 ^$LOCK

^$L[OCK] (expr V nref)

^$LOCK will provide information on the existence and operational characteristics of locked nrefs.

7.1.3.8 ^$ROUTINE

^$R[OUTINE] (routinexpr)

routinexpr ::= expr V routinename

^$ROUTINE provides information about the existence and characteristics of routines.

If and only if a routine identified by routinexpr exists, ^$ROUTINE(routinexpr) is defined ($DATA returns a non-zero value); all non-empty string values are reserved for future extensions to the standard. Process characteristics are stored beneath the ^$ROUTINE(routinexpr) node:

^$ROUTINE(routinexpr , expr V "CHARACTER") = charsetexpr

This node identifies the Character Set Profile in which routine routinexpr is stored.

When a routine is created and ^$ROUTINE(routinexpr ,"CHARACTER") for that routine has a $DATA value of zero, then this node is assigned the current value of the node ^$JOB($JOB, "CHARACTER").

7.1.3.9 ^$SYSTEM

^$S[YSTEM] (systemexpr)

systemexpr ::= expr V system

system ::= syntax of $SYSTEM intrinsic special variable

^$SYSTEM provides information about the characteristics of systems. A system represents the domain of concurrent processes for which $JOB is unique; the current system is identified by the svn $SYSTEM. The second level subscripts of ^$SYSTEM not beginning with the letter "Z" are reserved for future extensions to the standard.

7.1.3.9.1 Characteristic: Localized Formatting

If an implementation provides the function $%FORMAT^STRING (see page 75), the following nodes may be defined.

^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”DC”) = $expr_3$
^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”EC”) = $expr_3$
^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”FS”) = $expr_3$
^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”FM”) = $expr_3$
^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”SL”) = $expr_3$
^$SYSTEM (systemexpr , $expr_1$ V ”FORMAT” , $expr_2$ V ”SR”) = $expr_3$

The values of these nodes will provide process-wide defaults for the various localized formatting parameters. Possible values for these nodes and their meanings are described on page 75.

7.1.3.9.1 System Character Set Profile

^$SYSTEM(systemexpr , expr V "CHARACTER") = charsetexpr

This node specifies the charset which the specified system uses for interpretation of all system-wide name values (syntactic elements, e.g. ssvn names, commandwords, svn names, et cetera). Note that this allows an implementation to provide $Z[*] names, et cetera, which include idents other than those in any standardized character profile.

7.1.3.9.2 System Collation Algorithm

^$SYSTEM(systemexpr , $expr_1$ V "COLLATE") = $expr_2$ V algoref

This node identifies the collation algorithm which the specified system uses for determining collation order for system syntactic elements.

7.1.3.10 ^$Y[unspecified]

These ssvns are reserved for users and are called user-defined structured system variables. The syntax is

^ $ Y name [(L expr)]

An implementation is required to provide a means of associating a routine with one or more specific user-defined structured system variables.

This routine is called whenever a reference is made to a user-defined structured system variable by calling one of the following labels in the routine with first the parameter being the value of $NAME of the reference and the second parameter, if any, as below:

Reference Type	Label	Second Parameter	Result?
Evaluation	%VALUE		Yes
$DATA parameter	%DATA		Yes
$GET parameter	%GET	Second parameter of $GET	Yes
$ORDER paramater	%ORDER	Second parameter of $ORDER	Yes
$QUERY parameter	%QUERY	Second parameter of $QUERY	Yes
KILL command	%KILL		No
KSUBSCRIPTS command	%KSUBSCRIPTS		No
KVALUE command	%KVALUE		No
MERGE command target	%MERGE	Source glvn	No
MERGE command source	%MERGES	Target glvn	No
Value assignment	%SET	Value to be assigned	No

The usage of $ORDER and $QUERY with unsubscripted user-defined system variables has the same effect as if they were not user defined.

Editor's note:
This seems erroneous to me: $QUERY of an unsubscripted reference is no different from $QUERY of a subscripted reference, it just should return the namevalue of the first data node. I think it should not be labeled as an exception here.
$ORDER of an unsubscripted reference is not defined in this standard.

If both the source and target of a MERGE command are user-defined structured system variables, then only the MERGE label for the target ssvn is called.

Let *r* be the name of the routine being called when a reference is made to a certain user-defined structured system variable, and let *l* be the label at which this routine is called. If the routine does not exist when a reference is made to the user-defined structured system variable, then an error condition occurs with ecode = M97. If the routine exists, but the label does not exist when a reference is made to the user-defined structured system variable, then an error condition occurs with ecode = M13.

Note: names of user-defined structured system variables which differ only in the use of corresponding upper and lower case letters are not equivalent.

Note: Users providing routines to implement user-defined structured system variables are responsible for ensuring that other side-effects (such as a change to $TEST or $DATA values), which would not have taken place, had the reference been to a global variable, do not occur as a result of calling the routine.

7.1.3.11 ^$Z[unspecified]

^$Z[unspecified] (unspecified)

^$Z[unspecified] will provide implementation-specific information. Z is the initial letter for defining non-standard structured system variables. The requirement that ^$Z be used permits the unused initial letters to be reserved for future extensions to the standard without altering the execution of existing programs which observe the rules of the standard.

7.1.4 Expression item expritem

expritem ::= | strlit | numlit | exfunc | exvar | svn | function | unaryop expratom | (expr) |

7.1.4.1 String literal strlit

strlit ::= " ["" | nonquote] ... "

nonquote ::= any of the characters in graphic except the quote character

In words, a string literal is bounded by quotes and contains any string of printable characters, except that when quotes occur inside the string literal, they occur in adjacent pairs. Each such adjacent quote pair denotes a single quote in the value denoted by strlit, whereas any other printable character between the bounding quotes denotes itself. An empty string is denoted by exactly two quotes.

7.1.4.2 Numeric literal numlit

The integer literal intlit is a non-empty string of digits.

intlit ::= digit ...

The numeric literal numlit is defined as follows.

numlit ::= mant [exp]

mant ::= | intlit [. intlit] | . intlit |

exp ::= E [+ | !] intlit

The value of the string denoted by an occurrence of numlit is defined in the following two subclauses.

7.1.4.3 Numeric data values

The set of numeric values is a subset of the set of all values of data type MVAL. Only numbers that may be represented with a finite number of decimal digits are representable as numeric values. A data value of data type MVAL has the form of a number if it satisfies the following restrictions.

a. It shall contain only digits and the characters "!" and ".".

b. At least one digit must be present.

c. "." occurs at most once.

d. The number zero is represented by the one-character string "0".

e. The representation of each positive number contains no "!".

f. The representation of each negative number contains the character "!" followed by the representation of the positive number which is the absolute value of the negative number. (Thus, the following restrictions describe positive numbers only.)

g. The representation of each positive integer contains only digits and no leading zero.

h. The representation of each positive number less than 1 consists of a "." followed by a non-empty digit string with no trailing zero. (This is called a *fraction*.)

i. The representation of each positive non-integer greater than 1 consists of the representation of a positive integer (called the *integer part* of the number) followed by a fraction (called the *fraction part* of the number).

Note that the mapping between representable numbers and representations is one-to-one. An important result of this is that string equality of numeric values is a necessary and sufficient condition of numeric equality.

7.1.4.4 Meaning of numlit

Note that numlit denotes only non-negative values. The process of converting the spelling of an occurrence of numlit into its numeric data value consists of the following steps.

a. If the mant has no ".", place one at its right end.

b. If the exp is absent, skip step c.

c. If the exp has a plus or has no sign, move the "." a number of decimal digit positions to the right in the mant equal to the value of the intlit of exp, appending zeros to the right of the mant as necessary. If the exp has a minus sign, move the "." a number of decimal digit positions to the left in the mant equal to the value of the intlit of exp, appending zeros to the left of the mant as necessary.

d. Delete the exp and any leading or trailing zeros of the mant.

e. If the rightmost character is ".", remove it.

f. If the result is empty, make it "0".

7.1.4.5 Numeric interpretation of data

Certain operations, such as arithmetic, deal with the numeric interpretations of their operands. The numeric interpretation is a mapping from the set of all data values into the set of all numeric values, described by the following algorithm. Note that the numeric interpretation maps numeric values into themselves.

(Note: The *head* of a string is defined to be a substring which contains an identical sequence of characters in the string to the left of a given point and none of the characters in the string to the right of that point. A head may be empty or it may be the entire string.)

Consider the argument to be the string *S*.

First, apply the following sign reduction rules to *S* as many times as possible, in any order.

a. If *S* is of the form + *T*, then remove the +. (Shorthand: + *T* Y *T*)

b. ! + *T* Y ! *T*

c. ! ! *T* Y *T*

Second, apply one of the following, as appropriate.

a. If the leftmost character of *S* is not "!", form the longest head of *S* which satisfies the syntax description of numlit. Then apply the algorithm of 7.1.4.4 to the result.

b. If *S* is of the form ! *T*, apply step a) above to *T* and append a "!" to the left of the result. If the result is "!0", change it to "0".

The *numeric expression* numexpr is defined to have the same syntax as expr. Its presence in a syntax description serves to indicate that the numeric interpretation of its value is to be taken when it is executed.

numexpr ::= expr

7.1.4.6 Integer interpretation

Certain functions deal with the integer interpretations of their arguments. The integer interpretation is a mapping from the set of all data values onto the set of all integer values, described by the following algorithm.

First, take the numeric interpretation of the argument. Then remove the fraction, if present. If the result is empty or "!", change it to "0".

The *integer expression* intexpr is defined to have the same syntax as expr. Its presence in a syntax definition serves to indicate that the integer interpretation of its value is to be taken when it is executed.

intexpr ::= expr

7.1.4.7 Truth-value interpretation

The truth-value interpretation is a mapping from the set of all data values onto the two integer values 0 (false) and 1 (true), described by the following algorithm. Take the numeric interpretation. If the result is not "0", make it "1".

The *truth-value expression* tvexpr is defined to have the same syntax as expr. Its presence in a syntax definition serves to indicate that the truth-value interpretation of its value is to be taken when it is executed.

tvexpr ::= expr

7.1.4.8 Extrinsic function exfunc

exfunc ::= $ | $ labelref
libraryref
externref | actuallist

Extrinsic functions invoke a subroutine to return a value. When an extrinsic function is executed, the current value of $TEST, the current execution level, and the current execution location are saved in an exfunc frame on the PROCESS-STACK. The actuallist parameters are then processed as described in 8.1.7.

Execution continues either in the specified externref or at the first command of the formalline specified by the labelref. This formalline must contain a formallist in which the number of names is greater than or equal to the number of names in the actuallist, otherwise an error condition occurs with ecode = "M58". Execution of an exfunc to a levelline causes an error condition with ecode = "M20".

Upon return from the subroutine the value of $TEST and the execution level are restored, and the value of the argument of the QUIT command that terminated the subroutine is returned as the value of the exfunc.

7.1.4.9 Extrinsic variable exvar

exvar ::= $ | $ labelref
libraryref
externref |

An extrinsic special variable whose labelref is *x* is identical to the extrinsic function:

$$*x*( )

Note that label *x* must have a (possibly empty) formallist.

7.1.4.10 Intrinsic special variable names svn

Intrinsic special variables are denoted by the prefix $ followed by one of a designated list of names. Intrinsic special variable names differing only in the use of corresponding upper and lower case letters are equivalent. The standard contains the following intrinsic special variable names:

D[EVICE]
EC[ODE]
ES[TACK]
ET[RAP]
H[OROLOG]
I[O]

IOR[EFERENCE]
J[OB]
K[EY]
PIOR[EFERENCE]
P[RINCIPAL]
Q[UIT]
R[EFERENCE]
ST[ACK]
S[TORAGE]
SY[STEM]
T[EST]
TL[EVEL]
TR[ESTART]
X
Y
Z[unspecified]

Unused intrinsic special variable names beginning with an initial letter other than Z are reserved for future extensions to the standard.

The formal definition of the syntax of svn is a choice from among all of the individual svn syntax definitions of this subclause.

svn ::=

syntax of $DEVICE intrinsic special variable
syntax of $IO intrinsic special variable
...
syntax of $Y intrinsic special variable
syntax of $Z[unspecified] intrinsic special variable

Any implementation of the language must be able to recognize both the abbreviation and the full spelling of each intrinsic special variable name.

7.1.4.10.1 $DEVICE

$D[EVICE]

reflects the status of the current device. If the status of the device does not reflect anysignificant change of status, the value of $DEVICE, when interpreted as a truth-value, will be 0 (false). If there would have been a significant change to the status of the device, the value of $DEVICE, when interpreted as a truth-value, will be 1 (true). When a process is initiated, but before any commands are processed, the value of $DEVICE is an empty string if $IO is given a value which is the empty string, otherwise it is given an implementation-dependent value.

$DEVICE will give status code and meaning in one access. Its value is one of

M
M,I
M,I,T

where M is an MDC defined value , I is an implementor defined value and T is explanatory text.

The value of M, when interpreted as a truth value, will be equal to 0 (zero) when no significant change of status is being reported. Any non-zero value indicates a significant change of status.

The value of I is an implementation-specific value for the relevant status-information.

The value of T is implementation specific.

Note: Since M, I, and T are separated by commas, the values of M and I cannot contain this character.

7.1.4.10.2 $ECODE

$EC[ODE]

contains information about an error condition. When the value of $ECODE is the empty string, normal routine execution rules are in effect. When $ECODE contains anything else, the execution rules in 6.3.2 (Error processing) are active. When a process is initiated, but before any commands are processed, the value of $ECODE is the empty string.

The syntax of a non-empty value returned by $ECODE is as follows:

, L ecode ,

ecode ::= | M
U
Z | [noncomma ...]

noncomma ::= any of the characters in graphic except the comma character

Note: ecodes beginning with:

M	are reserved for the MDC
U	are reserved for the user
Z	are reserved for the implementation

All other values are reserved.

When an attempt is made to assign a value to $ECODE that does not meet the above constraints, an error will occur with ecode = “M101".

7.1.4.10.3 $ESTACK

$ES[TACK]

counts stack levels in the same way as $STACK, however, a NEW $ESTACK saves the value of $ESTACK and then assigns $ESTACK the value of 0. When a process is initiated, but before any commands are processed, the value of $ESTACK is 0 (zero).

7.1.4.10.4 $ETRAP

$ET[RAP]

contains code which is invoked in the event an error condition occurs. See 6.3.2- Error processing. When a process is initiated, but before any commands are processed, the value of $ETRAP is the empty string.

The value of $ETRAP may be stacked with the NEW command; NEW $ETRAP has the effect of saving the current instantiation of $ETRAP and creating a new instantiation initialized with the same value.

The value of $ETRAP is changed with the SET command. Changing the value of $ETRAP with a SET command instantiates a new trap; it does not save the old trap.

A QUIT from $ETRAP, either explicit or implicit (i.e., SET $ETRAP = "DO ^ETRAP" has an implicit QUIT at its end with an empty argument, if appropriate) will function as if a QUIT had been issued at the "current" $STACK. Behavior at the "popped" level will be determined by the value of $ECODE. If $ECODE is empty, execution proceeds normally. Otherwise, $ETRAP is invoked at the new level.

7.1.4.10.5 $HOROLOG

$H[OROLOG]

gives date and time with one access. Its value is *D* , *S* where *D* is an integer value counting days since an origin specified below, and *S* is an integer value modulo 86,400 counting seconds. The value of $HOROLOG for the first second of December 31, 1840 is defined to be 0,0. *S* increases by 1 each second and *S* clears to 0 with a carry into *D* on the tick of midnight.

7.1.4.10.6 $IO

$I[O]

identifies the current I/O device (see 8.2.7 and 8.2.35). Its value has the form of expr. When a process is initiated, but before any commands are processed, the value of $IO is equal to the value of $PRINCIPAL if an implicit OPEN and USE for the device specified by $PRINCIPAL is executed by the implementation. If the implementation does not execute these OPEN and USE commands then $IO is given the value of the empty string.

7.1.4.10.7 $IOREFERENCE

$IOR[EFERENCE]

identifies the current I/O device (see 8.2.7 and 8.2.35). Its value has the syntax of devn with the following restrictions:

a. When a process is initiated, but before any commands are processed, the value of $IOREFERENCE is equal to the value of $PRINCIPAL if an implicit OPEN and USE for the device specified by $PRINCIPAL is executed by the implementation. If the implementation does not execute these OPEN and USE commands then $IOREFERENCE is given the value of the empty string.

b. If the last command that changed $IOREFERENCE included an environment, then the value returned by $IOREFERENCE shall include that environment; otherwise the value of $IOREFERENCE shall not include an environment.

c. An environment whose value has the form of a number as defined in 7.1.4.3 appears as a numlit, spelled as its numeric interpretation.

d. An environment whose value does not have the form of a number as defined in 7.1.4.3 appears as a strlit.

7.1.4.10.8 $JOB

$J[OB]

Each executing process has its own job number, a positive integer which is the value of $JOB. The job number of each process is unique to that process within a domain of concurrent processes defined by the implementor (see also $SYSTEM; the concatenation of $SYSTEM and $JOB uniquely identifies a process within the universe of all M[UMPS] processes). $JOB is constant throughout the active life of a process.

7.1.4.10.9 $KEY

$K[EY]

contains the control-sequence which terminated the last READ command from the current device (including any introducing and terminating characters). If no READ command was issued to the current device or when no terminator was used, the value of $KEY will be the empty string. The effect of a READ *glvn on $KEY is unspecified. When a process is initiated, but before any commands are processed, the value of $KEY is an empty string if $IO is given a value which is the empty string, otherwise it is given an implementation-dependent value.

If a Character Set Profile input-transform is in effect, then this transformation is also applied to the value stored in $KEY. Certain mnemonicspaces may also specify that $KEY contains values as a result of other I/O commands.

See (READ command) and (WRITE command).

7.1.4.10.10 $PIOREFERENCE

$PIOR[EFERENCE]

identifies the principal I/O device. When a process is initiated, but before any commands are processed, the value of $PIOREFERENCE is equal to the value of $PRINCIPAL with the following restrictions:

a. If $PRINCIPAL is the empty string, then $PIOREFERENCE is the empty string.

b. If $PRINCIPAL is not the empty string, then $PIOREFERENCE shall include an environment.

7.1.4.10.11 $PRINCIPAL

$P[RINCIPAL]

identifies the principal I/O device, which is defined in the following fashion:

a. If the process is initiated by another M[UMPS] process then $PRINCIPAL is given the value of $PRINCIPAL of the initiating process, unless overriden by implementation-specific JOB parameters.

b. If the process is initiated from a specific device then $PRINCIPAL is given the identifier of that device.

c. Otherwise $PRINCIPAL is given an implementation-specific value.

$PRINCIPAL is constant throughout the active life of a process.

7.1.4.10.12 $QUIT

$Q[UIT]

returns 1 if the current PROCESS-STACK frame was invoked by an exfunc or exvar, and therefore a QUIT would require an argument. Otherwise, $QUIT returns 0 (zero). When a process is initiated, but before any commands are processed, the value of $QUIT is 0 (zero).

7.1.4.10.13 $REFERENCE

$R[EFERENCE]

returns the namevalue of the most recently referenced gvn, on which the current value of the naked indicator is based; for the behavior after a reference to the function $QUERY see 7.1.5.15. When a process is initiated, but before any commands are processed, the value of $REFERENCE is an empty string.

The value of $REFERENCE may be set to either the empty string, or to a namevalue, indicating a gvn. A side-effect of setting $REFERENCE equal to the empty string is that the naked indicator will become undefined. A side-effect of setting $REFERENCE to a namevalue is that the naked indicator will change as if the indicated gvn had been referenced.

7.1.4.10.14 $STACK

$ST[ACK]

gives the current level of the PROCESS-STACK. $STACK contains an integer value of zero or greater. When a process is initiated, but before any commands are processed, the value of $STACK is 0 (zero). See 7.1.2.3 (process-stack) for a description of stack behavior.

7.1.4.10.15 $STORAGE

$S[TORAGE]

Each implementation must return for the value of $STORAGE an integer which is the number of characters of free space available for use. The method of arriving at the value of $STORAGE is not part of the standard.

7.1.4.10.16 $SYSTEM

$SY[STEM]

Each implementation must return a value in $SYSTEM which represents uniquely the system representing the domain of concurrent processes for which $JOB is unique. Its value is *V,S* where *V* is an integer value allocated by the MDC to an implementor and *S* is defined by that implementor in such a way as to be able to be unique for all the implementor's systems.

7.1.4.10.17 $TEST

$T[EST]

contains the truth value computed from the execution of an IF command containing an argument, or an OPEN, LOCK, JOB, or READ command with a timeout (see 7.1.4.8, 7.1.4.9, and 8.2.8). When a process is initiated, but before any commands are processed, the value of $TEST is 0 (false).

7.1.4.10.18 $TLEVEL

$TL[EVEL]

indicates whether a TRANSACTION is currently in progress. When a process is initiated, but before any commands are processed, the value of $TLEVEL is 0. TSTART adds 1 to $TLEVEL. When $TLEVEL is greater than 0, TCOMMIT subtracts 1 from $TLEVEL. A ROLLBACK or RESTART sets $TLEVEL to 0.

7.1.4.10.19 $TRESTART

$TR[ESTART]

indicates how many RESTARTs have occurred since the initiation of a TRANSACTION. When a process is initiated, but before any commands are processed, the value of $TRESTART is 0, and it is set to 0 by the successful completion of TCOMMIT or TROLLBACK. Each RESTART adds 1 to $TRESTART.

7.1.4.10.20 $X

$X

has a non-negative integer value which approximates the value of the horizontal co-ordinate of the active position on the current device. It is set to zero by any control-function or format that involves a move to the start of a line. When a process is initiated, but before any commands are processed, the value of $X is 0 if $IO is given a value which is the empty string, otherwise it is given an implementation-dependent value.

The unit in which $X is expressed is initially equal to 'characters'. Certain formats may change this.

When any control-function would leave the cursor in a position so that the horizontal co-ordinate would be uncertain, the value of $X will not be changed. In such cases the value of $DEVICE will reflect this status.

If a Character Set Profile input-transform is in effect, then $X is modified in accordance with the input prior to any transform taking place. If a Character Set Profile output-transform is in effect, then $X is modified in accordance with the output after any transform takes place.

See 8.2.27 (READ command) 8.2.35 (USE command) and 8.2.37 (WRITE command).

7.1.4.10.21 $Y

$Y

has a non-negative integer value which approximates the value of the vertical co-ordinate of the active position on the current device. It is set to zero by any control-function or format that involves a move to the start of a page. When a process is initiated, but before any commands are processed, the value of $Y is 0 if $IO is given a value which is the empty string, otherwise it is given an implementation-dependent value.

The unit in which $Y is expressed is initially equal to 'lines'. Certain formats may change this.

When any control-function would leave the cursor in a position so that the vertical co-ordinate would be uncertain, the value of $Y will not be changed. In such cases, the value of $DEVICE will reflect this status.

If a Character Set Profile input-transform is in effect, then $Y is modified in accordance with the input prior to any transform taking place. If a Character Set Profile output-transform is in effect, then $Y is modified in accordance with the output after any transform takes place.

See 8.2.27 (READ command) 8.2.35 (USE command) and 8.2.37 (WRITE command).

7.1.4.10.22 $Z

$Z[unspecified]

Z is the initial letter reserved for defining non-standard intrinsic special variables. The requirement that $Z be used permits the unused initial letters to be reserved for future extensions to the standard without altering the execution of existing routines which observe the rules of the standard.

7.1.4.11 Unary operator unaryop

unaryop ::= | ‘ | (Note: apostrophe)
| + |
| ! | (Note: hyphen)

There are three unary operators: ' (not), + (plus), and ! (minus).

Not inverts the truth value of the expratom immediately to its right. The value of 'expratom is 1 if the truth-value interpretation of expratom is 0; otherwise its value is 0. Note that '' performs the truth-value interpretation.

Plus is merely an explicit means of taking a numeric interpretation. The value of +expratom is the numeric interpretation of the value of expratom.

Minus negates the numeric interpretation of expratom. The value of !expratom is the numeric interpretation of !*N*, where *N* is the value of expratom.

Note that the order of application of unary operators is right-to-left.

7.1.4.12 Name value namevalue

namevalue ::= expr

A namevalue has the syntax of a glvn with the following restrictions:

a. The glvn is not a naked reference.

b. Each subscript whose value has the form of a number appears as specified in 7.1.4.3.

c. Each subscript whose value does not have the form of a number as defined in 7.1.4.3 appears as a sublit, defined as follows:

sublit ::= “ | “” | ... “
| subnonquote |

where subnonquote is defined as follows:

subnonquote ::= any character valid in a subscript, excluding the quote symbol

d. The environment appears as defined in b. and c. for subscripts.

e. If the glvn is an ssvn, the name part of the ssvn will appear in uppercase in the unabbreviated form.

7.1.5 Intrinsic function function

Intrinsic functions are denoted by the prefix $ followed by one of a designated list of names, followed by a parenthesized argument list. Intrinsic function names differing only in the use of corresponding upper and lower case letters are equivalent. The following function names are defined:

functionname ::=	A[SCII] C[HAR] D[ATA] E[XTRACT] F[IND] FN[UMBER] G[ET] H[OROLOG] J[USTIFY] L[ENGTH] NA[ME] O[RDER] P[IECE] QL[ENGTH] QS[UBSCRIPT] Q[UERY] R[ANDOM] RE[VERSE] S[ELECT] ST[ACK] T[EXT] TY[PE] TR[ANSLATE] V[IEW] Z[unspecified]

Unused function names beginning with an initial letter other than Z are reserved for future extensions to the standard.

The formal definition of the syntax of function is a choice from among all of the individual function syntax definitions in this subclause.

function ::=	syntax of $ASCII function syntax of $CHAR function ... syntax of $VIEW function syntax of $Z[unspecified] function

Any implementation of the language must be able to recognize both the abbreviation and the full spelling of each function name.

7.1.5.1 $ASCII

$A[SCII] (expr)

This form produces an integer value as follows:

a. ! 1 if the value of expr is the empty string.

b. Otherwise, an integer *n* associated with the leftmost character of the value of expr, such that $ASCII($CHAR(*n*)) = *n*.

$A[SCII] (expr , intexpr)

This form is similar to $ASCII(expr) except that it works with the intexprth character of expr instead of the first. Formally, $ASCII(expr,intexpr) is defined to be $ASCII($EXTRACT(expr,intexpr)).

7.1.5.2 $CHAR

$C[HAR] (L intexpr)

This form returns a string whose length is the number of argument expressions which have non-negative values. Each intexpr in the closed interval [0,127] maps into the ASCII character whose code is the value of intexpr; this mapping is order-preserving. Each negative-valued intexpr maps into no character in the value of $CHAR. Each intexpr greater than 127 maps into a character in a manner defined by the current charset of the process.

7.1.5.3 $DATA

$D[ATA] (glvn)

This form returns a non-negative 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 which satisfies the following conditions:

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

b. the first *N* descriptors 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.

7.1.5.4 $DEXTRACT

$DE[XTRACT] ([initialrecordvalue] , extracttemplate , L [recordfieldvalue])

initialrecordvalue ::= expr

extracttemplate ::= expr V extractfields

extractfields ::= L | [!] fieldwidth [: fieldindex] |

fieldwidth ::= intlit

fieldindex ::= intlit

recordfieldvalue ::= expr [: fieldindex]

This function assembles the exprs of the recordfieldvalue into a single value. The value of the initialrecordvalue is used as the starting value to which the recordfieldvalues are applied. If initialrecordvalue is omitted, the empty string is used.

The extracttemplate specifies the $EXTRACT partitioning (fieldwidths and alignments) of initialrecordvalue into consecutive fields. Unsigned values specify width for left-justified fields. Negative values specify width (absolute value of fieldwidth) for right-justified fields. The fieldindex specifies the relative field number. If omitted, it defaults to the next successive value. For omitted recordfieldvalues the corresponding field is obtained from the initialrecordvalue. Although all elements of the list of recordfieldvalues are optional, at least one recordfieldvalue (not necessarily the first) in the list must be non-empty.

Left-justified fields are padded on the right with $CHAR(32) or truncated on the right as needed. Right-justified fields are padded on the left with $CHAR(32) or truncated on the left as needed.

Assignment to fields proceeds in a left-to-right fashion. If a field is referenced multiple times, the rightmost value is the final value of the field.

7.1.5.5 $DPIECE

$DP[IECE] ([initialrecordvalue] , piecedelimiter , L [recordfieldvalue])

piecedelimiter ::= expr

This function assembles the exprs of the recordfieldvalues into a single value. The value of initialrecordvalue is used as the starting value to which the recordfieldvalues are applied. If initialrecordvalue is omitted, the empty string is used. The piecedelimiter specifies the relative field number. If omitted, it defaults to the next successive value. For omitted recordfieldvalues the corresponding field is obtained from the initialrecordvalue. Although all elements of the list of recordfieldvalues are optional, at least one recordfieldvalue (not necessarily the first) in the list must be non-empty.

7.1.5.6 $EXTRACT

$E[XTRACT] (expr)

This form returns the first (leftmost) character of the value of expr. If the value of expr is the empty string, the empty string is returned.

$E[XTRACT] (expr , intexpr)

Let *s* be the value of expr, and let *m* be the integer value of intexpr. $EXTRACT(*s*,*m*) returns the *m*th character of *s*. If *m* is less than 1 or greater than $LENGTH(*s*), the value of $EXTRACT is the empty string. (1 corresponds to the leftmost character of *s*; $LENGTH(*s*) corresponds to the rightmost character.)

$E[XTRACT] (expr , intexpr_1 , intexpr_2)

Let *n* be the integer value of intexpr_2. $EXTRACT(*s*,*m*,*n*) returns the string between positions *m* and *n* of *s*. The following cases are defined:

a. *m* > *n*
   Then the value of $EXTRACT is the empty string.

b. *m* = *n*
   $EXTRACT(*s*,*m*,*n*) = $EXTRACT(*s*,*m*).

c. *m* < *n* '> $LENGTH(*s*)
   $EXTRACT(*s*,*m*,*n*) = $EXTRACT(*s*,*m*) concatenated with $EXTRACT(*s*,*m*+1,*n*).
   That is, using the concatenation operator _ of 7.2.1.1,
   $EXTRACT(*s*,*m*,*n*) = $EXTRACT(*s*,*m*)_$EXTRACT(*s*,*m*+1)_..._$EXTRACT(*s*,*m*+(*n*!*m*)).

d. *m* < *n* and $LENGTH(*s*) < *n*
$EXTRACT(*s*,*m*,*n*) = $EXTRACT(*s*,*m*,$LENGTH(*s*)).

7.1.5.7 $FIND

$F[IND] (expr$_1$, expr$_2$)

This form searches for the leftmost occurrence of the value of expr$_2$ in the value of expr$_1$. If none is found, $FIND returns zero. If one is found, the value returned is the integer representing the number of the character position immediately to the right of the rightmost character of the found occurrence of expr$_2$ in expr$_1$. In particular, if the value of expr$_2$ is empty, $FIND returns 1.

$F[IND] (expr$_1$, expr$_2$, intexpr)

Let *a* be the value of expr$_1$, let *b* be the value of expr$_2$, and let *m* be the value of intexpr. $FIND(*a*,*b*,*m*) searches for the leftmost occurrence of *b* in *a*, beginning the search at the max(*m*,1) position of *a*. Let *p* be the value of the result of $FIND($EXTRACT(*a*,*m*,$LENGTH(*a*)),*b*). If no instance of *b* is found (i.e., *p*=0), $FIND returns the value 0; otherwise, $FIND(*a*,*b*,*m*) = *p* + max(*m*,1) ! 1.

7.1.5.8 $FNUMBER

$FN[UMBER] (numexpr , fncodexpr)

fncodexpr ::= expr V fncode

fncode ::= [fncodatom ...]

fncodatom ::= fncodp | fncodt | , | + | ! (note, comma) (note, hyphen)

fncodp ::= p | P

fncodt ::= t | T

This form shall return a value that is the value of numexpr edited by applying each fncodatom according to the following rules. The order of application is not significant:

fncodatom	Action
fncodp	Represent negative numexpr values in parentheses. Let *A* be the absolute value of numexpr. Use of fncodp will result in the following: 1) If numexpr < 0, the result will be "("_*A*_")". 2) If numexpr ‘< 0, the result will be " "_*A*_" ".
fncodt	Represent numexpr with a trailing rather than a leading "+" or "!" sign. Note: if sign suppression is in force (either by default on positive values,

or by design using the "!" fncodatom), use of fncodt will result in a trailing space character.

, Insert comma delimiters every third position to the left of the decimal (present or assumed) within numexpr. Note: no comma shall be inserted which would result in a leading comma character.

\+ Force a plus sign ("+") on positive values of numexpr. Position of the "+" (leading or trailing) is dependent on whether or not fncodt is present.

! Suppress the negative sign "!" on negative values of numexpr.

All other values for fncodatom are reserved. Note: Zero is neither positive nor negative.

If fncodexpr equals an empty string, no special formatting is performed and the result of the expression is the original value of numexpr.

More than one occurrence of a particular fncodatom within a single fncode is identical to a single occurrence of that fncodatom. An error condition occurs, with ecode = "M2", when a fncodp is present with any of the sign suppression or sign placement fncodatoms ("+!" or fncodt).

Note: if (!1 < numexpr < 1), the result of $FNUMBER(numexpr,fncodexpr) does not have a leading zero (“0") to the left of the decimal point.

$FN[UMBER] (numexpr , fncodexpr , intexpr)

This form is identical to the two-argument form of $FNUMBER, except that numexpr is rounded to intexpr fraction digits, including possible trailing zeros, before processing any fncodatoms. If intexpr is zero, the evaluated numexpr contains no decimal point. Note: if (!1 < numexpr < 1), the result of this form of $FNUMBER has a leading zero ("0") to the left of the decimal point. Negative values of intexpr are reserved for future extensions to the $FNUMBER function.

7.1.5.9 $GET

$G[ET] (glvn)

This form returns the value of the specified glvn depending on its state, defined by $DATA(glvn). The following cases are defined:

a. $DATA(glvn) # 10 = 1
   The value returned is the value of the variable specified by glvn.

b. Otherwise, the value returned is the empty string.

$G[ET] (glvn , expr)

This form returns the value of the specified glvn depending on its state, defined by $DATA(glvn). The following cases are defined:

a. $DATA(glvn) # 10 = 1
   The value returned is the value of the variable specified by glvn.

b. Otherwise, the value returned is the value of expr.

Both glvn and expr will be evaluated before the function returns a value, so that the behavior of this function with respect to the naked indicator is well defined.

7.1.5.10 $HOROLOG

$H[OROLOG] (intexpr)

This form gives date, time, and time-offset with one access. Let *m* be the value of intexpr, and let:

D be an integer counting days since the origin used by $HOROLOG. (Like the *D* in $HOROLOG).

S be a numeric value counting seconds since local midnight. (Like *S* in $HOROLOG but not restricted to integer values if the M[UMPS] implementation can supply it).

C be a numeric value counting the number of seconds since the origin used by $HOROLOG, but not restricted to integer values.

TZ be the number of seconds in the time-offset needed to get UCT (Greenwich Mean Time) from local time (local time + *TZ* = UCT).

The following cases are defined:

a. If *m* = 0, this function returns a string in the format "*D,S,TZ*", where *S* is restricted to an integer value. (The value of $HOROLOG with offset).

b. If *m* = 1, this function returns a string for local time in the format "*D,S,TZ*", where *S* is not restricted to an integer value.

c. If *m* = !1, this function returns a string for UCT in the format "*D,S,TZ*", where *S* is not restricted to an integer value.

d. All other values of *m* are reserved.

7.1.5.11 $JUSTIFY

$J[USTIFY] (expr , intexpr)

This form returns the value of expr right-justified in a field of intexpr spaces. Let *m* be $LENGTH(expr) and *n* be the value of intexpr. The following cases are defined:

a. *m* '< *n*
Then the value returned is expr.

b. Otherwise, the value returned is *S*(*n* ! *m*) concatenated with $expr_1$, where *S*(*x*) is a string of *x* spaces.

$J[USTIFY] (numexpr , $intexpr_1$, $intexpr_2$)

This form returns an edited form of the number numexpr. Let *r* be the value of numexpr after rounding to $intexpr_2$ fraction digits, including possible trailing zeros. (If $intexpr_2$ is the value 0, *r* contains no decimal point.) The value returned is $JUSTIFY(*r*, $intexpr_1$). Note that if !1 < numexpr < 1, the result of $JUSTIFY does have a zero to the left of the decimal point. Negative values of $intexpr_2$ are reserved for future extensions to the $JUSTIFY function.

7.1.5.12 $LENGTH

$L[ENGTH] (expr)

This form returns an integer which is the number of characters in the value of expr. If the value of expr is

the empty string, $LENGTH(expr) returns the value 0.

$L[ENGTH] ($expr_1$, $expr_2$)

This form returns the number plus one of non-overlapping occurrences of $expr_2$ in $expr_1$. If the value of $expr_2$ is the empty string, then $LENGTH returns the value 0.

7.1.5.13 $MUMPS

$M[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:

L 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 that 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 non-positive values of intlit are reserved for future extensions to the standard.

7.1.5.14 $NAME

$NA[ME] (glvn)

This form returns a string value which is the namevalue denoting the named glvn. Note that naked references are permitted in the argument, but that the returned value is always a non-naked reference. If glvn includes an environment, then the namevalue shall include the canonic representation of that environment; otherwise the namevalue shall not include an environment.

$NA[ME] (glvn , intexpr)

This form returns a string value which is a namevalue denoting either all or part of the supplied glvn, depending on the value of intexpr. Let $NAME(glvn) applied to the supplied glvn be of the form Name(s_1, s_2, ..., s_n), considering *n* to be zero if the glvn has no subscripts, and let *m* be the value of intexpr. Then $NAME(glvn , intexpr) is defined as follows:

a. It is erroneous for *m* to be less than zero (ecode = "M39").

b. If *m* = 0, the result is Name.

c. If *n* > *m*, the function returns the string returned by $NAME(Name($s_1$, s_2, ..., s_m)).

d. Otherwise, the function returns the string returned by $NAME(glvn).

7.1.5.15 $ORDER

$O[RDER] (glvn)

This form returns a value which is a subscript according to a subscript ordering sequence. This ordering sequence is specified below with the aid of a function, CO, which is used for definitional purposes only, to establish the collating sequence.

CO(*s*,*t*) is defined, for strings *s* and *t*, as follows:

> When *t* follows *s* in the ordering sequence or if *s* is the empty string, CO(*s*,*t*) returns *t*. Otherwise, CO(*s*,*t*) returns *s*.

The ordering sequence is defined using the *collation algorithm* determined as follows:

a. If $ORDER refers to an ssvn, then the algorithm is determined by the value of ^$SYSTEM($SYSTEM, "COLLATE"); if that node does not exist, then the value of
$GET(^$CHARACTER(^$SYSTEM,"CHARACTER"),"COLLATE"))
is used.

b. If $ORDER refers to a gvn with name *^global* then the algorithm is determined by the value of ^$GLOBAL("*^global* ", "COLLATE"); if that node does not exist, then the value of
$GET(^$CHARACTER(^$GLOBAL("*^global* ","CHARACTER"),"COLLATE"))
is used.

c. If $ORDER does not refer to either of the above, then the algorithm is determined by the value of $GET(^$CHARACTER(^$JOB($JOB,"CHARACTER"),"COLLATE")).

d. If the resulting algorithm is the empty string, then the *collation algorithm* of the charset M (defined in Annex A) is used.

The collation value *order* of a string *subscript* using a collation algorithm *collate* may be determined by executing the expression ("S *order*="_*collate*_"(*subscript*)"). Two collation values are compared on a character-by-character basis using the $ASCII values (i.e. equivalent to the follows (]) operator).

Only subscripted forms of glvn are permitted. Let glvn be of the form NAME(s_1, s_2, ..., s_n) where s_n may be the empty string. Let *A* be the set of subscripts such that, *s* is in *A* if and only if:

a. CO(s_n,*s*) = *s* and

b. $DATA(NAME($s_1$, s_2, ..., s_{n-1}, *s*)) is not zero.

Then $ORDER(NAME($s_1$, s_2, ..., s_n)) returns that value *t* in *A* such that CO(*t*,*s*) = *s* for all *s* not equal to *t*; that is, all other subscripts in *A* which follow s_n also follow *t*.

If no such *t* exists, $ORDER returns the empty string.

$O[RDER] (glvn , expr)

Let S be the value of expr. Then $ORDER(glvn,expr) returns:

a. If S = 1, the function returns a result identical to that returned by $ORDER(glvn).

b. If S = !1, the function returns a value which is a subscript, according to a subscript ordering sequence. This ordering sequence is specified below with the aid of functions CO and CP, which are

used for definitional purposes only, to establish the collating sequence.

CO(*s*,*t*) is defined, for strings *s* and *t*, according to the *collation algorithm* of the specific charset.

CP(*s*,*t*) is defined, for strings *s* and *t*, as follows:

When *t* follows *s* in the ordering sequence and *s* is not the empty string, CP(*s*,*t*) returns *s*. Otherwise, CP(*s*,*t*) returns *t*.

The following cases define the ordering sequence for CP:

1. CP("",*t*) = *t*.
2. CP(*s*,*t*) = *t* if CO(*s*,*t*) = *s*; otherwise, CP(*s*,*t*) = *s*.

Only subscripted forms of glvn are permitted. Let glvn be of the form NAME(s_1, s_2, ..., s_n) where s_n may be the empty string. Let A be the set of subscripts such that, *s* is in *A* if and only if:

1. CP(s_n, *s*) = *s* and
2. $DATA(NAME($s_1$, s_2, ..., $s_{n!1}$, *s*)) is not zero.

Then $ORDER(NAME($s_1$, s_2, ..., s_n), !1) returns that value *t* in *A* such that CP(*t*,*s*) = *t* for all *s* not equal to *t*; that is, all other subscripts in *A* which precede *s* also precede *t*.

If no such *t* exists, $ORDER(NAME($s_1$, s_2, ..., s_n), !1) returns the empty string.

c. Values of S other than 1 and !1 are reserved for future extensions to the $ORDER function.

7.1.5.16 $PIECE

$P[IECE] ($expr_1$, $expr_2$)

This form is defined here with the aid of a function, NF, which is used for definitional purposes only, called *find the position number following the* m*th occurrence*.

NF(*s* , *d* , *m*) is defined, for strings *s*, *d*, and integer *m*, as follows:

When *d* is the empty string, the result is zero.

When *m* '> 0, the result is zero.

When *d* is not a substring of *s*, i.e., when $FIND(*s* , *d*) = 0, then the result is $LENGTH(*s*) + $LENGTH(*d*) + 1.

Otherwise, NF(*s* , *d* , 1) = $FIND(*s* , *d*).

For *m* > 1, NF(*s* , *d* , *m*) =
NF($EXTRACT(*s* , $FIND(*s* , *d*), $LENGTH(*s*)), *d* , *m* !1) + $FIND(*s* , *d*) ! 1.

That is, NF extends $FIND to give the position number of the character to the right of the *m*th occurrence of the string *d* in *s*.

Let *s* be the value of $expr_1$, and let *d* be the value of $expr_2$. $PIECE(*s*,*d*) returns the substring of *s* bounded on the right but not including the first (leftmost) occurrence of *d*.

$PIECE(*s* , *d*) = $EXTRACT(*s* , 0, NF(*s* , *d* ,1) ! $LENGTH(*d*) ! 1).

$P[IECE] (expr$_1$, expr$_2$, intexpr)

Let *m* be the integer value of intexpr. $PIECE(*s*,*d*,*m*) returns the substring of *s* bounded by but not including the *m*!*1*th and the *m*th occurrence of *d*.

$PIECE(*s* , *d* , *m*) = $EXTRACT(*s* , NF(*s* , *d* , *m* !1), NF(*s* , *d* , *m*) ! $LENGTH(*d*) ! 1).

$P[IECE] (expr$_1$, expr$_2$, intexpr$_1$, intexpr$_2$)

Let *m* be the integer value of intexpr$_1$.Let *n* be the integer value of intexpr$_2$. $PIECE(*s* , *d* , *m* , *n*) returns the substring of *s* bounded on the left but not including the *m*!*1*th occurrence of *d* in *s*, and bounded on the right but not including the *n*th occurrence of *d* in *s*.

$PIECE(*s* , *d* , *m* , *n*) = $EXTRACT(*s* , NF(*s* , *d* , *m* !1), NF(*s* , *d* , *n*) ! $LENGTH(*d*) !1).

Note that $PIECE(*s* , *d* , *m* , *m*) = $PIECE(*s* , *d* , *m*), and that $PIECE(*s* , *d* ,1) = $PIECE(*s* , *d*).

7.1.5.17 $QLENGTH

$QL[ENGTH] (namevalue)

See 7.1.4.12 for the definition of namevalue.

This form returns a value which is derived from namevalue. If namevalue has the form NAME(s_1, s_2, ..., s_n), considering *n* to be zero if there are no subscripts, then the function returns *n*.

Note that the namevalue is not "executed", and will not affect the naked indicator, nor generate an error if the namevalue represents an undefined glvn. The naked indicator will only be affected by the last gvn reference (if any) executed while evaluating the parameter.

7.1.5.18 $QSUBSCRIPT

$QS[UBSCRIPT] (namevalue , intexpr)

This form returns a value which is derived from namevalue. If namevalue has the form NAME(s_1 , s_2 , ... , s_n), considering *n* to be zero if there are no subscripts, and *m* is the value of intexpr, then $QSUBSCRIPT(namevalue , intexpr) is defined as follows:

a. Values of *m* less than !1 are reserved for possible future extensions to the standard.

b. If *m* = !1, the result is the environment if namevalue includes an environment; otherwise the empty string.

c. If *m* = 0, the result is NAME without an environment even if one is present.

d. If *m* > *n*, the result is the empty string.

e. Otherwise, the result is the subscript value denoted by s_m.

Note that the namevalue is not "executed", and will not affect the naked indicator, nor generate an error if the namevalue represents an undefined glvn. The parameters are evaluated in left to right order, and the naked indicator will only be affected by the last gvn reference (if any) executed while evaluating them.

7.1.5.19 $QUERY

$Q[UERY] (glvn)

Follow these steps:

a. Let glvn be a variable reference of the form Name(s_1, s_2, ..., s_q) where s_q may be the empty string. If glvn is unsubscripted, initialize *V* to the form Name(""); otherwise, initialize *V* to glvn.

b. If the last subscript of *V* is empty, Goto step e.

c. If $DATA(*V*) \ 10 = 1, append the subscript "" to *V*, i.e., *V* is Name(s_1, s_2, ..., s_q, "").

d. If *V* has no subscripts, return "".

e. Let *s* = $ORDER(*V*).

f. If *s* = "", truncate the last subscript off *V*, Goto step d.

g. If *s* '= "", replace the last subscript in *V* with *s*.

h. If $DATA(*V*) # 2 = 1, return *V* formatted as a namevalue.

i. Goto step c.

$Q[UERY] (glvn , expr)

Let S be the value of expr. Then $QUERY(glvn , expr) returns:

1. If S = 1, the function returns a result identical to that returned by $QUERY(glvn).

2. If S = !1, the function returns a value which is either the empty string ("") or a namevalue according to the following steps:

    a. Let glvn be a variable reference of the form Name(s_1 , s_2 , ... , s_q) where s_q may be the empty string. If glvn is unsubscripted, initialize *V* to the form Name(""); otherwise, initialize *V* to glvn.

    b. If the last subscript of *V* is empty, go to step e.

    c. If $DATA(*V*)\10=1, append the subscript "" to *V*, i.e. *V* is Name(s_1 , s_2 , ... , s_q ,"").

    d. If *V* has no subscripts, return "".

    e. Let *s* = $ORDER(*V* , !1).

    f. If *s* = "", truncate the last subscript off *V*, and go to step j.

    g. If *s* '= "", replace the last subscript of *V* with *s*.

    h. If $DATA(*V*) # 2 = 1, return *V* formatted as namevalue.

    i. Go to step c.

    j. If $DATA(*V*) # 2 = 1, return *V* formatted as a namevalue

k. Go to step d.

3. Values of S other than 1 or !1 are reserved for future extensions to the $QUERY function.

If the value of $QUERY(glvn [, expr]) is not the empty string, and glvn includes an environment, then the namevalue shall include the environment; otherwise the namevalue shall not include an environment.

If the argument of $QUERY is a gvn, the naked indicator will become undefined and the value of $REFERENCE will become equal to the empty string.

7.1.5.20 $RANDOM

$R[ANDOM] (intexpr)

This form returns a random or pseudo-random integer uniformly distributed in the closed interval [0, intexpr ! 1]. If the value of intexpr is less than 1, an error condition occurs with ecode = "M3".

7.1.5.21 $REVERSE

$RE[VERSE] (expr)

See Clause 7 for the definition of expr.

This form returns a string whose characters are reversed in order compared to expr.

$REVERSE(EXPR) is computationally equivalent to $$REV(EXPR) which is defined by the following code:

```
REV(E) Q $SELECT(E="":"",1:$$REV($EXTRACT(E,2,$LENGTH(E)))_$EXTRACT(E,1))
```

Editor's note:
Since the above recursive algorithm cannot be executed on a system that conforms exactly to the portability limitations (120 DO levels, strings longer than 120 characters), suggest to change this to:

```
REV(E) New I,R
 Set R="" For I=1:1:$Length(E) Set R=$Extract(E,I)_R
 Quit R
```

7.1.5.22 $SELECT

$S[ELECT] (L * tvexpr : expr *)

This form returns the value of the leftmost expr whose corresponding tvexpr is true. The process of evaluation consists of evaluating the tvexprs, one at a time in left-to-right order, until the first one is found whose value is true. The expr corresponding to this tvexpr (and no other) is evaluated and this value is made the value of $SELECT. An error condition occurs, with ecode = "M4", if all tvexprs are false. Since only one expr is evaluated at any invocation of $SELECT, that is the only expr which must have a defined value.

7.1.5.23 $STACK

$ST[ACK] (intexpr)

This form returns a string as follows:

a. If intexpr is !1:

1. If $ECODE is not empty, returns the highest value where $STACK(intexpr) can return non-empty results.

2. If $ECODE is empty then return $STACK.

b. If intexpr is 0 (zero), returns an implementation specific value indicating how this process was started.

c. If intexpr is greater than 0 (zero) and less than or equal to $STACK indicates how this level of the PROCESS-STACK was created:

1. If due to a command, the commandword, fully spelled out and in uppercase.
2. if due to an exfunc or exvar, the string "$$".
3. if due to an error, the ecode representing the error that created the result returned by $STACK(intexpr).

d. If intexpr is greater than $STACK, returns an empty string.

e. Values of intexpr less than -1 are reserved for future extensions to the $STACK function.

$ST[ACK] (intexpr , stackcodexpr)

stackcodexpr ::= expr V stackcode

stackcode ::= “PLACE” | “MCODE” | “ECODE”

This form returns information about the action that created the level of the PROCESS-STACK identified by intexpr as follows:

a. Values of intexpr < 0 are reserved.

b. Values of intexpr > $STACK return the empty string.

stackcode	**Returned String**
ECODE	the list of any ecodes added at the level of the PROCESS-STACK identified by intexpr.
MCODE	the value (in the case of an XECUTE) or the line for the location identified by $STACK(intexpr , "PLACE"). If the line is not available, an empty string is returned.
PLACE	the location of a command at the intexpr level of the PROCESS-STACK as follows: a. if intexpr is not equal to $STACK and $STACK(intexpr , "ECODE") would return the empty string, the last command executed. b. if intexpr is equal to $STACK and $STACK(intexpr , "ECODE") would return the empty string, the currently executing command. c. if $STACK(intexpr , "ECODE") would return a non-empty string, the last command to start execution while $STACK(intexpr , "ECODE") would have returned the empty string. The location is in the form:

place SP + eoffset

place ::= | [label] [+ intlit] [^ VB environment VB routinename] |
@

eoffset ::= intlit

In place, the first case is used to identify the line being executed at the time of creation of this level of the PROCESS-STACK. The second case (@) shows the point of execution occurring in an XECUTE.

eoffset is an offset into the code or data identified by place at which the error occurred. The value might point to the first or last character of a "token" just before or just after a "token", or even to the command or line in which the error occurred. Implementors should provide as accurate a value for eoffset as practical.

All values of stackcode beginning with the letter Z are reserved for the implementation. All other values of stackcode are reserved for future extensions to the $STACK function. stackcodes differing only in the use of corresponding upper and lower case letters are equivalent.

7.1.5.24 $TEXT

$T[EXT] (textarg)

textarg ::= | + intexpr [^ routineref] |
entryref |
@ expratom V textarg |

This form returns a string whose value is the contents of the line specified by the parameter. Specifically, the entire line, with eol deleted, is returned.

If the argument of $TEXT is an entryref, the line denoted by the entryref is specified. If entryref does not contain dlabel then the line denoted is the first line of the routine. If the argument is of the form + intexpr [^ routineref], two cases are defined. If the value of intexpr is greater than 0, the intexprth line of the routine is specified; if the value of intexpr is equal to 0, the routinename of the routine is specified. An error condition occurs, with ecode = "M5", if the value of intexpr is less than 0. In all cases, if no routine is explicitly specified, the currently-executing routine is used.

If no such line as that specified by the parameter exists, an empty string is returned. If the line specification is ambiguous, the results are not defined.

If a Character Set Profile input-transform is in effect, then the string is modified in accordance with the transform.

7.1.5.25 $TYPE

$TY[PE] (expratom)

This form returns the string value "MVAL" if the value of the expratom is not a value of data type OREF. Otherwise, $TYPE returns the string "OBJECT".

7.1.5.26 $TRANSLATE

$TR[ANSLATE] ($\underline{expr}_1$, $\underline{expr}_2$)

Let *s* be the value of $\underline{expr}_1$, $TRANSLATE($\underline{expr}_1$, $\underline{expr}_2$) returns an edited form of *s* in which all characters in *s* which are found in $\underline{expr}_2$ are removed.

$TR[ANSLATE] ($\underline{expr}_1$, $\underline{expr}_2$, $\underline{expr}_3$)

Let *s* be the value of $\underline{expr}_1$, $TRANSLATE($\underline{expr}_1$, $\underline{expr}_2$, $\underline{expr}_3$) returns an edited form of *s* in which all characters in *s* which are found in $\underline{expr}_2$ are replaced by the positionally corresponding character in $\underline{expr}_3$. If a character in *s* appears more than once in $\underline{expr}_2$ the first (leftmost) occurrence is used to positionally locate the translation.

Translation is performed once for each character in *s*. Characters which are in *s* that are not in $\underline{expr}_2$ remain unchanged. Characters in $\underline{expr}_2$ which have no corresponding character in $\underline{expr}_3$ are deleted from *s* (this is the case when $\underline{expr}_3$ is shorter than $\underline{expr}_2$).

Note: If the value of $\underline{expr}_2$ is the empty string, no translation is performed and *s* is returned unchanged.

7.1.5.27 $VIEW

$V[IEW] (unspecified)

makes available to the implementor a call for examining machine-dependent information. It is to be understood that routines containing occurrences of $VIEW may not be portable.

7.1.5.28 $Z

$Z[unspecified] (unspecified)

is the initial letter reserved for defining non-standard intrinsic functions. This requirement permits the unused function names to be reserved for future use.

7.1.6 M[UMPS] Standard Library

7.1.6.1 Library definitions

A library consists of a set of libraryelements - functions and data which are accessed from M[UMPS] and which have unique names within the library. The access method for each libraryelement is the external calling syntax, which normally has no side-effects.

A library is defined as being either mandatory or optional. library names starting with a Z are reserved for implementors. library names starting with a Y are reserved for users. All other unused library names are reserved for future use.

The M[UMPS] Standard Library is the set of library definitions in this standard.

The following librarys are defined:

7.1.6.1.1 Mandatory Libraries

CHARACTER
MATH
STRING

7.1.6.1.2 Optional Libraries

None defined at this time.

7.1.6.2 Library Element Definitions

The definition of a libraryelement states which library the element belongs to, return value type, and full specification.

libraryelement names starting with a Z are reserved for implementors. libraryelement names starting with a Y are reserved for users. All other unused libraryelement names are reserved for future use.

A libraryelement definition is of the form:

libraryelementdef ::= libraryelement ^ library libraryresult [(L libraryparam)]

libraryparam ::= [.] name [: [libdatatype] [: libraryopt]]

libraryresult ::= [: libdatatype]

libdatatype ::=

BOOLEAN
COMPLEX
INTEGER
MATRIX
NAME
REAL
STRING
Z[unspecified]

libraryopt ::=

o
O

If a libraryparam starts with a period then this parameter is passed by reference.

Z is the initial letter reserved for implementation specific libdatatypes. All other values for libdatatypes are reserved for future extensions to the standard.

Input and output values to libraryelements undergo the appropriate data interpretation below:

a. For BOOLEAN see 7.1.4.7 (Truth-value interpretation).

b. COMPLEX numbers are represented as strings of the format REAL_"%"_REAL (that is, two REAL numbers separated by the % character). Any string has a value when interpreted as a complex number. The real part of the complex number is the numeric interpretation of the first "%" piece and the imaginary part is the numeric interpretation of the second "%" piece. The canonic representation of a complex number is a string created by concatenating the canonic numerical representation of the real part, a percent sign, and the canonic numerical representation of the imaginary part.

c. For INTEGER see 7.1.4.6 (Integer interpretation).

d. MATRIX values are represented as sparse arrays, in which only the defined nodes with two integer-valued subscripts will be accessed or modified. Any other nodes in the arrays are not considered part of the matrix, and do not affect and are not affected by the matrix functions. The string *A[R,C]* denotes a matrix *A* having *R* rows and *C* columns. The canonic representation of a matrix value

contains no non-matrix array nodes.

e. For NAME see 7.1.4.12 (Name value namevalue).

f. For REAL see 7.1.4.5 (Numeric interpretation of data).

g. STRING is a string made up of any characters and not constrained in format.

If no libdatatype is specified for a libraryparam or libraryresult then the libdatatype defaults to STRING.

If no libraryopt is specified then the libraryparam is mandatory. A libraryopt of O specifies that the libraryparam is optional.

Unless otherwise specified in their definitions, library elements are assumed to have the standard domain and range for their function, and no side effects. For each library element with a standard domain, *all* of its libraryparams can assume any valid values of their respective libdatatypes. Similarly, for each function with a standard range, the libraryresult and *all* of its output libraryparams can assume any valid values of their respective libdatatypes.

7.1.6.3 Availability of library elements

An implementation of M[UMPS] shall

a. provide the mandatory librarys defined in this standard

and

b. provide a means by which replacement definitions in routines of libraryelements can be installed so that a routine can access them as if they were part of the implementation. An implementation may additionally provide a means by which non-M[UMPS] code can be installed to implement libraryelements.

An implementation may also provide a means by which specific librarys or libraryelements of the M[UMPS] Standard Library are only optionally installed.

7.1.6.4 CHARACTER Library elements

7.1.6.4.1 $%COLLATE^CHARACTER

COLLATE^CHARACTER : STRING (A : STRING , CHARMOD : : O)

$%COLLATE^CHARACTER returns the collation value of a string according to the specification of the collation algorithm. CHARMOD is either a Character Set Profile specification in the form charset or a global variable name specification in the form ^name. If CHARMOD is a Character Set Profile then the collation algorithm used is that specified in ^$CHARACTER for that profile. If CHARMOD is a global variable name then the collation algorithm used is that specified in ^$GLOBAL for that name. If CHARMOD is not specified, or the node specified above does not exist, then the collation algorithm used is the default process collating algorithm.

7.1.6.4.2 $%COMPARE^CHARACTER

COMPARE^CHARACTER : INTEGER (A : STRING , B : STRING , CHARMOD : : O)

$%COMPARE^CHARACTER compares two strings according to the specification of the collation algorithm, and returns:

! 1 if A collates before B

0 if A collates the same as B
1 if A collates after B

CHARMOD is either a Character Set Profile specification in the form charset or a global variable name specification in the form ^name. If CHARMOD is a Character Set Profile then the two strings are compared using the collation algorithm specified in ^$CHARACTER for that profile. If CHARMOD is a global variable name then the two strings are compared using the collation algorithm specified in ^$GLOBAL for that name. If CHARMOD is not specified, or the node specified above does not exist, then the two strings are compared using the default process collation algorithm.

7.1.6.4.3 $%LOWER^STRING

LOWER^STRING : STRING (A : STRING , CHARMOD : O)

$%LOWER^STRING returns a string that is an edited version of the value of its first parameter, in which all upper-case characters are converted to the corresponding lower-case characters.

Editor's note:
It would seem to be that this function is a function that relates to ^$CHARACTER more than to strings in general. Suggest to specify it as $%LOWER^CHARACTER.

If the value of CHARMOD is a namevalue referencing a gvn, then the conversion algorithm used is that specified in ^$GLOBAL for that gvn. If the value of CHARMOD is a charsetexpr, then the conversion algorithm used is that specified in ^$CHARACTER for that character set profile. If CHARMOD is not specified, or the node specified above does not exist, then the conversion algorithm used is that specified as the default for the process in ^$JOB.

If no algorithm is specified in the appropriate ssvn, then the characters A through Z are converted to a through z respectively.

If CHARMOD references a gvn, it must be either of the form ^ name or of the form ^ VB environment VB name.

7.1.6.4.4 $%PATCODE^STRING

PATCODE^STRING : BOOLEAN (A : STRING , PAT : STRING , CHARMOD : O)

$%PATCODE^STRING returns a tvexpr that indicates whether or not the value passed as its first parameter matches the specified patcode in the specified character set.

Editor's note:
It would seem to be that this function is a function that relates to ^$CHARACTER more than to strings in general. Suggest to specify it as $%PATCODE^CHARACTER.

If the value of CHARMOD is a namevalue referencing a gvn, then the conversion algorithm used is that specified in ^$GLOBAL for that gvn. If the value of CHARMOD is a charsetexpr, then the conversion algorithm used is that specified in ^$CHARACTER for that character set profile. If CHARMOD is not specified, or the node specified above does not exist, then the conversion algorithm used is that specified as the default for the process in ^$JOB.

If CHARMOD references a gvn, it must be either of the form ^ name or of the form ^ VB environment VB name

7.1.6.4.5 $%UPPER^STRING

UPPER^STRING : STRING (A : STRING , CHARMOD : O)

$%UPPER^STRING returns a string that is an edited version of the value of its first parameter, in which all lower-case characters are converted to the corresponding upper-case characters.

Editor's note:
It would seem to be that this function is a function that relates to ^$CHARACTER more than to strings in general. Suggest to specify it as $%UPPER^CHARACTER.

If the value of CHARMOD is a namevalue referencing a gvn, then the conversion algorithm used is that specified in ^$GLOBAL for that gvn. If the value of CHARMOD is a charsetexpr, then the conversion algorithm used is that specified in ^$CHARACTER for that character set profile. If CHARMOD is not specified, or the node specified above does not exist, then the conversion algorithm used is that specified as the default for the process in ^$JOB.

If no algorithm is specified in the appropriate ssvn, then the characters a through z are converted to A through Z respectively.

If CHARMOD references a gvn, it must be either of the form ^ name or of the form ^ VB environment VB name.

7.1.6.5 MATH Library elements

7.1.6.5.1 $%ABS^MATH

ABS^MATH : REAL (X : REAL)

$%ABS^MATH returns the absolute value of its parameter.

7.1.6.5.2 $%ARCCOS^MATH

ARCCOS^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCOS^MATH returns the value of the trigonometric arccosine, expressed in radians, of X; 0 # $%ARCCOS^MATH(X) # B. The number of significant digits in the arccosine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X < !1 or X > 1, an error condition occurs with ecode = "M28".

7.1.6.5.3 $%ARCCOSH^MATH

ARCCOSH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCCOSH^MATH returns the value of the hyperbolic arccosine, expressed in radians, of X; $%ARCCOSH^MATH(X) $ 0. The number of significant digits in the hyperbolic arccosine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X < 1, an error condition occurs with ecode = "M28".

7.1.6.5.4 $%ARCCOT^MATH

ARCCOT^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCCOT^MATH returns the value of the trigonometric arccotangent, expressed in radians, of X; 0 < $%ARCCOT^MATH(X) < B. The number of significant digits in the arccotangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.5 $%ARCCOTH^MATH

ARCCOTH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCCOTH^MATH returns the value of the hyperbolic arccotangent, expressed in radians, of X; $%ARCCOTH^MATH(X) < 0 when X # !1, and $%ARCCOTH^MATH(X) >0 when X $ 1. The number of significant digits in the hyperbolic arccotangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When !1 < X < 1, an error condition occurs with ecode = "M28".

7.1.6.5.6 $%ARCCSC^MATH

ARCCSC^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCCSC^MATH returns the value of the trigonometric arccosecant, expressed in radians, of X; 0 # $%ARCCSC^MATH(X) # B. The number of significant digits in the arccosecant is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X < !1 or X > 1, an error condition occurs with ecode = "M28".

7.1.6.5.7 $%ARCSEC^MATH

ARCSEC^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCSEC^MATH returns the value of the trigonometric arcsecant, expressed in radians, of X; 0 # $%ARCSEC^MATH(X) # B. The number of significant digits in the arcsecant is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X < !1 or X > 1, an error condition occurs with ecode = "M28".

7.1.6.5.8 $%ARCSIN^MATH

ARCSIN^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCSIN^MATH returns the value of the trigonometric arcsine, expressed in radians, of X; -B / 2 # $%ARCSIN^MATH(X) # B / 2. The number of significant digits in the arcsine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X < !1 or X > 1, an error condition occurs with ecode = "M28".

7.1.6.5.9 $%ARCSINH^MATH

ARCSINH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCSINH^MATH returns the value of the hyperbolic arcsine, expressed in radians, of X. The number of significant digits in the hyperbolic arcsine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.10 $%ARCTAN^MATH

ARCTAN^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCTAN^MATH returns the value of the trigonometric arctangent, expressed in radians, of X; |$%ARCTAN^MATH(X) | # B / 2, 0 # $%ARCTAN^MATH(X) # B when X $ 0, and !B # $%ARCTAN^MATH(X) # 0 when X # 0. The number of significant digits in the arctangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.11 $%ARCTANH^MATH

ARCTANH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%ARCTANH^MATH returns the value of the hyperbolic artangent, expressed in radians, of X. The number of significant digits in the hyperbolic artangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X # !1 or X $ 1, an error condition occurs with ecode = "M28".

7.1.6.5.12 $%CABS^MATH

CABS^MATH : REAL (Z : REAL)

$%CABS^MATH returns the absolute value of the complex number Z.

7.1.6.5.13 $%CADD^MATH

CADD^MATH : COMPLEX (X : COMPLEX , Y : COMPLEX)

$%CADD^MATH returns the sum of X + Y, where X and Y are complex numbers.

7.1.6.5.14 $%CCOS^MATH

CCOS^MATH : COMPLEX (Z : COMPLEX , PREC : INTEGER : O)

$%CCOS^MATH returns the value of the trigonometric cosine cos(Z) of the angle Z. The value of Z is expressed in radians. !1 # $%CCOS^MATH(Z) # 1. Z is interpreted as a complex number. The number of significant digits in the complex cosine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.15 $%CDIV^MATH

CDIV^MATH : COMPLEX (X : COMPLEX , Y : COMPLEX)

$%CDIV^MATH returns the value X / Y, where X and Y are complex numbers. If the complex numeric interpretation of Y is equal to "0%0", an error condition occurs with ecode = "M9"

7.1.6.5.16 $%CEXP^MATH

CEXP^MATH : COMPLEX (Z : COMPLEX , PREC : INTEGER : O)

$%CEXP^MATH returns the value of *e* raised to the power of the complex number Z. The number of significant digits in the complex exponent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.17 $%CLOG^MATH

CLOG^MATH : COMPLEX (Z : COMPLEX , PREC : INTEGER : O)

$%CLOG^MATH returns the logarithm of the complex number Z; Re ($%CLOG^MATH(Z)) can be any number, -B # Im ($%CLOG^MATH(Z)) # B. The number of significant digits in the complex logarithm is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. If Im (Z) = 0, then Re (Z) > 0.

7.1.6.5.18 $%CMUL^MATH

CMUL^MATH : COMPLEX (X : COMPLEX , Y : COMPLEX)

$%CMUL^MATH returns the value of X * Y, where X and Y are complex numbers.

7.1.6.5.19 $%COMPLEX^MATH

COMPLEX^MATH : COMPLEX (X : REAL)

$%COMPLEX^MATH returns the complex representation of the number specified in X.

7.1.6.5.20 $%CONJUG^MATH

CONJUG^MATH : COMPLEX (Z : COMPLEX)

$%CONJUG^MATH returns the value of the conjugate of the complex number Z.

7.1.6.5.21 $%COS^MATH

COS^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%COS^MATH returns the value of the trigonometric cosine of X. The value of X is expressed in radians. !1 # $%COS^MATH(X) # 1. The number of significant digits in the cosine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.22 $%COSH^MATH

COSH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%COSH^MATH returns the value of the hyperbolic cosine of X. The value of X is expressed in radians. $%COSH^MATH(X) $ 1. The number of significant digits in the hyperbolic cosine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.23 $%COT^MATH

COT^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%COT^MATH returns the value of the trigonometric cotangent of X. The value of X is expressed in radians. The number of significant digits in the cotangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.24 $%COTH^MATH

COTH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%COTH^MATH returns the value of the hyperbolic cotangent of X. The value of X is expressed in radians. $%COTH^MATH(X) < !1 when X < 0 and $%COTH^MATH(X) > 1 when X > 0. The number of significant digits in the hyperbolic cotangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

Editor's note:
Recommend to add to this paragraph:

If the value of X is equal to zero, an error condition occurs with ecode = "M28".

(Or should that be error code M9 ?)

7.1.6.5.25 $%CPOWER^MATH

CPOWER^MATH : COMPLEX (Z : COMPLEX , X : COMPLEX , PREC : INTEGER : O)

$%CPOWER^MATH returns the value of Z raised to the power of X, where Z and X are complex numbers. The number of significant digits in the complex power is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.26 $%CSC^MATH

CSC^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%CSC^MATH returns the value of the trigonometric cosecant of X. The value of X is expressed in radians. $%CSC^MATH(X) # !1 or $%CSC^MATH(X) $ 1. The number of significant digits in the cosecant is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.27 $%CSCH^MATH

CSCH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%CSCH^MATH returns the value of the hyperbolic cosecant of X. The value of X is expressed in radians. The number of significant digits in the hyperbolic cosecant is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.28 $%CSIN^MATH

CSIN^MATH : COMPLEX (Z : COMPLEX , PREC : INTEGER : O)

$%CSIN^MATH returns the value of the trigonometric sine sin(Z) of Z. The value of Z is expressed in radians and is interpreted as a complex number; !1 # Re ($%CSIN^MATH(Z)) # 1, !1 # Im ($%CSIN^MATH(Z)) # 1. The number of significant digits in the complex sine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.29 $%CSUB^MATH

CSUB^MATH : COMPLEX (X : COMPLEX , Y : COMPLEX)

$%CSUB^MATH returns the value of X - Y, where X and Y are complex numbers.

7.1.6.5.30 $%DECDMS^MATH

DECDMS^MATH : STRING (X : REAL , PREC : INTEGER : O)

$%DECDMS^MATH returns a string, containing the °ℕO notation for the angle that is specified in X in degrees. Since the symbols for degrees, minutes, and seconds are not in the ASCII set, the fields in the result-value are separated by colons (":"); the value of the first part is an integer in the range [0,359]; the value in the second part is an integer in the range [0,59]; the value in the third part is a real number in the

range [0,60). The optional parameter PREC specifies the precision to which X is rounded before the conversion takes place. If not specified, a default value of 5 digits is assumed for PREC.

7.1.6.5.31 $%DEGRAD^MATH

DEGRAD^MATH : REAL (X : REAL)

$%DEGRAD^MATH returns the value in radians that is equal to the angle specified in X in degrees. A full circle is 2B radians, or 360 degrees.

7.1.6.5.32 $%DMSDEC^MATH

DMSDEC^MATH : REAL (X : STRING)

$%DMSDEC^MATH returns the value in degrees that is equal to the angle specified in X in °ℕ0 notation. Since the symbols for degrees, minutes, and seconds are not in the ASCII set, the three fields in X must be separated by colons (":"); the value of the first part is an integer in the range [0,359]; the value in the second part is an integer in the range [0,59]; the value in the third part is a real number in the range [0,60). Any further ":" separated parts in the value of X are ignored.

7.1.6.5.33 $%E^MATH

E^MATH : REAL ( )

$%E^MATH returns the value of *e* (Euler's number), approximated to at least 15 significant digits.

7.1.6.5.34 $%EXP^MATH

EXP^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%EXP^MATH returns the value of *e* (Euler's number) to the power of X. The exponentiation is approximated with as many significant digits as specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.35 $%LOG^MATH

LOG^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%LOG^MATH returns the Naperian logarithm of X. The number of significant digits in the logarithm is specified by the optional parameter PREC. If not supplied, a default value of 11 digits is assumed for PREC. When X is less than or equal to 0, an error condition occurs with ecode = "M28".

7.1.6.5.36 $%LOG10^MATH

LOG10^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%LOG10^MATH returns the Briggsian logarithm of X. The number of significant digits in the logarithm is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X is less than or equal to 0, an error condition occurs with ecode = "M28".

7.1.6.5.37 $%MTXADD^MATH

MTXADD^MATH : BOOLEAN (.A : MATRIX , .B : MATRIX , .R : MATRIX , ROWS : INTEGER , COLS : INTEGER)

$%MTXADD^MATH adds matrix B[ROWS,COLS] to matrix A[ROWS,COLS], and stores the result into

matrix R[ROWS,COLS]. It is permissible that the actual parameter for matrix R is equal to either of the actual parameters for matrices A and B. The return value is 1 if both matrices A and B exist; or 0 if a) there are no defined values in one or both of the matrices A and B, or b) when ROWS < 1 or COLS < 1.

7.1.6.5.38 $%MTXCOF^MATH

MTXCOF^MATH : REAL (.A : MATRIX , I : INTEGER , K : INTEGER , N : INTEGER)

$%MTXCOF^MATH computes the cofactor in matrix A[N,N] for element A(I,K). The return value is the value of the cofactor.

7.1.6.5.39 $%MTXCOPY^MATH

MTXCOPY^MATH : BOOLEAN (.A: MATRIX , .R: MATRIX , ROWS : INTEGER , COLS : INTEGER)

$%MTXCOPY^MATH copies the matrix A[ROWS,COLS] into the matrix R[ROWS,COLS]. The return value is 1 if matrix A exists; or 0 if a) there are no defined values in the matrix A, or b) when ROWS < 1 or COLS < 1.

7.1.6.5.40 $%MTXDET^MATH

MTXDET^MATH : REAL (.A : MATRIX , N : INTEGER)

$%MTXDET^MATH computes the determinant of matrix A[N,N]. The return value is the value of the determinant; or “” (empty) if a) the determinant cannot be computed, or b) when N < 1.

7.1.6.5.41 $%MTXEQU^MATH

MTXEQU^MATH : BOOLEAN (.A : MATRIX , .B : MATRIX , .R : MATRIX , N : INTEGER , M : INTEGER)

$%MTXEQU^MATH solves the matrix-equation A[M,N] * R[M,N] = B[M,N], with matrix R[M,N] being the unknown to be resolved. The return value is 1 if a solution to the equation can be computed, 0 if it cannot, or “” (the empty string) if M < 1 or N < 1.

The actual parameter for array R may not be equal to either of the references that are passed as the actual parameters for arrays A and B.

7.1.6.5.42 $%MTXINV^MATH

MTXINV^MATH : BOOLEAN (.A : MATRIX , .R : MATRIX , N : INTEGER)

$%MTXINV^MATH inverts matrix A[N,N] into matrix R[N,N]. It is permissible that the actual parameter for matrix R is equal to the actual parameter for matrix A. The return value is 1 if matrix
A has been inverted into matrix R; or 0 if a) no inverse matrix can be computed, or b) when N < 1.

7.1.6.5.43 $%MTXMUL^MATH

MTXMUL^MATH : BOOLEAN (.A : MATRIX , .B : MATRIX , .R : MATRIX , M : INTEGER , L : INTEGER , N : INTEGER)

$%MTXMUL^MATH multiplies matrix A[M,L] with matrix B[L,N]; the result is stored into matrix R[M,N]. The actual parameter for matrix R may not be equal to the actual parameter for matrix A, or the actual parameter for matrix B. The return value is 1 if both matrices A and B exist; or 0 if a) there are no defined values in one or both of the matrices A and B, or b) when L < 1 or M < 1 or N < 1.

7.1.6.5.44 $%MTXSCA^MATH

MTXSCA^MATH : BOOLEAN (.A : MATRIX , .R : MATRIX , ROWS : INTEGER , COLS : INTEGER , S : REAL)

$%MTXSCA^MATH multiplies scalar value S with matrix A[ROWS,COLS], and stores the result into matrix R[ROWS,COLS]. It is permissible that the actual parameter for matrix R is equal to the actual parameter for matrix A. The return value is 1 if matrix A exists; or 0 if a) there are no defined values in the matrix A, or b) when ROWS < 1 or COLS < 1.

7.1.6.5.45 $%MTXSUB^MATH

MTXSUB^MATH : BOOLEAN (.A : MATRIX , .B : MATRIX , .R : MATRIX , ROWS : INTEGER , COLS : INTEGER)

$%MTXSUB^MATH subtracts matrix B[ROWS,COLS] from matrix A[ROWS,COLS], and stores the result into matrix R[ROWS,COLS]. It is permissible that the actual parameter for matrix R is equal to either of the actual parameters for matrices A and B. The return value is 1 if both matrices A and B exist; or 0 if a) there are no defined values in one or both of the matrices A and B, or b) when ROWS < 1 or COLS < 1.

7.1.6.5.46 $%MTXTRP^MATH

MTXTRP^MATH : BOOLEAN (.A : MATRIX , .R : MATRIX , M : INTEGER , N : INTEGER)

$%MTXTRP^MATH transposes matrix A[M,N] into matrix R[N,M]. It is permissible that the actual parameter for matrix R is equal to the actual parameter for matrix A. The return value is 1 if matrix A exists; or 0 if a) there are no defined values in the matrix A, or b) when M < 1 or N < 1.

7.1.6.5.47 $%MTXUNIT^MATH

MTXUNIT^MATH : BOOLEAN (.R : MATRIX , N : INTEGER , SPARSE : BOOLEAN : O)

$%MTXUNIT^MATH creates matrix R[N,N] as a unit matrix. If the value of the optional parameter SPARSE is 1 (*true*), a sparse unit matrix will be created, i.e., only the diagonal elements of the result matrix will be defined. The return value is 1 if a unit matrix can be created; or 0 if a) it cannot be created, or b) when N < 1.

7.1.6.5.48 $%PI^MATH

PI^MATH : REAL ( )

$%PI^MATH returns the value of B (pi), approximated to at least 15 significant digits.

7.1.6.5.49 $%RADDEG^MATH

RADDEG^MATH : REAL (X : REAL)

%RADDEG^MATH returns the value in degrees that is equal to the angle specified in X in radians. A full circle is 2B radians, or 360 degrees.

7.1.6.5.50 $%SEC^MATH

SEC^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%SEC^MATH returns the value of the trigonometric secant of X. The value of X is expressed in radians. $%SEC^MATH(X) # !1 or $%SEC^MATH(X) $ 1. The number of significant digits in the secant is

specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.51 $%SECH^MATH

SECH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%SECH^MATH returns the value of the hyperbolic secant of X. The value of X is expressed in radians. 0 < $%SECH^MATH(X) # 1. The number of significant digits in the hyperbolic secant is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.52 $%SIGN^MATH

SIGN^MATH : REAL (X : REAL)

$%SIGN^MATH returns 0, !1, or 1, depending on the value of X. If X < 0, SIGN^MATH returns !1. If X equals 0, it returns 0. If X > 0, it returns 1.

7.1.6.5.53 $%SIN^MATH

SIN^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%SIN^MATH returns the value of the trigonometric sine of X. The value of X is expressed in radians. !1 # $%SIN^MATH(X) # 1. The number of significant digits in the sine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.54 $%SINH^MATH

SINH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%SINH^MATH returns the value of the hyperbolic sine of X. The value of X is expressed in radians. The number of significant digits in the hyperbolic sine is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.55 $%SQRT^MATH

SQRT^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%SQRT^MATH returns the square root of X. The number of significant digits in the square root is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC. When X is less than 0, an error condition occurs with ecode = "M28".

7.1.6.5.56 $%TAN^MATH

TAN^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%TAN^MATH returns the value of the trigonometric tangent of X. The value of X is expressed in radians. The number of significant digits in the tangent is specified by the optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.5.57 $%TANH^MATH

TANH^MATH : REAL (X : REAL , PREC : INTEGER : O)

$%TANH^MATH returns the value of the hyperbolic tangent of X. The value of X is expressed in radians. !1 # $%TANH^MATH(X) # 1. The number of significant digits in the hyperbolic tangent is specified by the

optional parameter PREC. If not specified, a default value of 11 digits is assumed for PREC.

7.1.6.6 STRING Library Elements

7.1.6.6.1 $%CRC16^STRING

CRC16^STRING : INTEGER (STRING : STRING , SEED : INTEGER : O)

$%CRC16^STRING computes a Cyclic Redundancy Code of the 8-bit character string STRING using $X^{16} + X^{15} + X^2 + 1$ as the polynomial. The optional parameter SEED supplies an initial value, which allows running CRC calculations on multiple strings. If missing, a default value of 0 (zero) is used for SEED. The message bytes are considered shifted in low order bit first and the return value shifted out low order bit first.

7.1.6.6.2 $%CRC32^STRING

CRC32^STRING : INTEGER (STRING : STRING , SEED : INTEGER : O)

$%CRC32^STRING computes a Cyclic Redundancy Code of the 8-bit character string STRING using $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + +X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ as the polynomial. The optional parameter SEED supplies an initial value, which allows running CRC calculations on multiple strings. If missing, a default value of 0 (zero) is used for SEED. The value of SEED is ones-complemented before being used. The message bytes are considered shifted in low order bit first and the return value is ones-complemented and shifted out low order bit first.

7.1.6.6.3 $%CRCCCITT^STRING

CRCCCITT^STRING : INTEGER (STRING : STRING , SEED : INTEGER : O)

$%CRCCCITT^STRING computes a Cyclic Redundancy Code of the 8-bit character string STRING using $X^{16} + X^{12} + X^5 + 1$ as the polynomial. The optional parameter SEED supplies an initial value, which allows running CRC calculations on multiple strings. If missing, a default value of 65535 (2^{16} ! 1) is used for SEED. The message bytes are considered shifted in high order bit first and the return value shifted out high order bit first.

7.1.6.6.4 $%FORMAT^STRING

FORMAT^STRING : STRING (IN : STRING , FORMAT : STRING)

fdirectives ::= expr V | fspec [: fspec] ... |

fspec ::= CS = expr | DC = fchar | EC = fchar | FS = expr | FM = fmask | SL = expr | SR = expr

fchar ::= expr V graphic

erchar ::= expr

fmask ::= expr V	c	...	Currency (left justified)
	d		Decimal (Singular occurrence per mask)
	f		Float either type at the end of the run
	l		Left Justified Numeric
	n		Numeric
	s		Separator
	x		Spacer
	!		Display sign only if negative
	+		Display sign always
	SP		Insert space for the spaces in the mask
	(		Start of negative value
	)		End of negative value

$%FORMAT^STRING returns a copy of IN formatted according to the instructions specified in FORMAT. The value of FORMAT is restricted to be an fdirectives. The meaning of these directives is explained below.

For each of the values below, the function will use the values that are specified. If, in a function call, there is no specification for any of the values below, a default value will be used. The function inherits its defaults from the process (see page 34, ^$JOB), and the process inherits its defaults from the system (see page 35, ^$SYSTEM).

FM = fmask
This specification provides the actual value for the format mask.
The format mask is a description of the various fields to be output. Should a sign not be specified (+, () or !), the absolute value of IN will be returned.

CS = expr
This specification provides the actual value for the currency string.
The currency string is a string of characters that represents either the single character local currency designator or the multiple character international reference. This string may occur only once in a format specification.

DC = fchar
This specification provides the actual value for the decimal separator character.
The decimal separator is the character that separates the units from the tenths in a numeric string.

EC = erchar
This specification provides the actual value for the error character.
The error character is repeated over the length of the result string when the formatting instructions cannot be applied.

FS = expr
This specification provides the actual value for the fill string.
The fill string is a repeating pattern to be used instead of spaces to fill unused columns.

SL = expr
This specification provides the actual value for the left-hand separator string.

SR = expr
This specification provides the actual value for the right-hand separator string.
The (left-hand and right-hand) separator strings are used to separate the various columns in positions that designate orders of magnitude. The left-hand separators are used for the positions to the left of the decimal separator; the right-hand separators are used for the positions to the right of the decimal separator.

The separator strings must contain one single character for each occurrence of a separator indicator in the format mask.

7.1.6.6.5 $%PRODUCE^STRING

PRODUCE^STRING : STRING (IN : STRING , .SPEC , MAX : INTEGER : O)

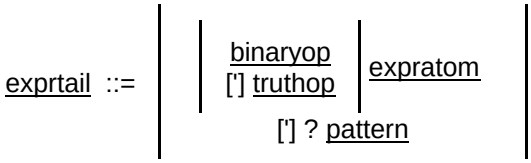
$%PRODUCE^STRING returns a copy of IN transformed by repeated prioritized substring replacement. For each element *e* in order by *e*'s collation sequence, let *find* be the value of SPEC(*e*,1) and let *out* be the value of SPEC(*e*,2). The function will scan IN for occurrences of the substring *find*. For each occurrence of *find* found in IN, whether or not any of the characters in the substring have already been replaced, the matching substring in IN is replaced with *out*. Processing of each substring is complete when no more occurrences of that substring can be found in IN; when all of the substrings in SPEC have been processed in order, the scanning and replacement begins again with the first *e* in SPEC; only when none of the substrings can be found is the resulting string is returned. The optional parameter MAX sets a maximum number of replacements $%PRODUCE^STRING will make before ending processing and returning the result.

7.1.6.6.6 $%REPLACE^STRING

REPLACE^STRING : STRING (IN : STRING , .SPEC)

$%REPLACE^STRING returns a copy of IN transformed by prioritized substring replacement. For each element *e* in order by *e*'s collation sequence, let *find* be the value of SPEC(*e*,1) and let *out* be the value of SPEC(*e*,2). The function will scan IN for occurrences of the substring *find*. For each occurrence of *find* found in IN, if none of the characters in that substring have already been replaced, then the matching substring in IN is replaced with OUT. Otherwise, the found substring is ignored. Processing of each substring is complete when no more unmodified occurrences of that substring can be found in IN; when all of the substrings in SPEC have been processed in order, the resulting string is returned.

7.2 Expression tail exprtail



The order of evaluation is as follows:

a. Evaluate the left-hand expratom.

b. If an exprtail is present immediately to the right, evaluate its expratom or pattern and apply its operator.

c. Repeat step b. as necessary, moving to the right.

In the language of operator precedence, this sequence implies that all binary string, arithmetic, and truth-valued operators are at the same precedence level and are applied in left-to-right order.

Any attempt to evaluate an expratom containing an lvn, gvn, ssvn, or svn with an undefined value is erroneous. A reference to a lvn with an undefined value causes an error condition with ecode = "M6". A reference to a gvn with an undefined value causes an error condition with ecode = "M7". A reference to an svn with an undefined value causes an error condition with ecode = "M8". A reference to an ssvn with an undefined value, where the semantics of that action are not specified for that specific ssvn, causes an error condition with ecode = "M60".

7.2.1 Binary operator binaryop

binaryop ::= | _ | (Note: underscore)
+
!
*
/
#
\
** | (Note: hyphen)

7.2.1.1 Concatenation operator

The underscore symbol _ is the concatenation operator. It does not imply any numeric interpretation. The value of *A_B* is the string obtained by concatenating the values of *A* and *B*, with *A* on the left.

7.2.1.2 Arithmetic binary operators

The binary operators + ! * / \ # ** are called the arithmetic binary operators. They operate on the numeric interpretations of their operands, and they produce numeric (in one case, integer) results.

+ produces the algebraic sum.

! produces the algebraic difference.

* produces the algebraic product.

/ produces the algebraic quotient. Note that the sign of the quotient is negative if and only if one operand is positive and one operand is negative. Division by zero causes an error condition with ecode = "M9".

\ produces the integer interpretation of the result of the algebraic quotient.

produces the value of the left operand modulo the right operand. It is defined only for non-zero values of its right operand, as follows.

A # *B* = *A* ! (*B* * floor(*A*/*B*))
where floor (*x*) = the largest integer '> *x*.
A value of 0 (zero) for *B* will produce an error condition with ecode = "M9".

** produces the exponentiated value of the left operand, raised to the power of the right operand. On an attempt to compute 0 * * (a negative number), an error condition occurs with ecode = "M9". On an attempt to compute 0 * * 0, an error condition occurs with ecode = "M94". On an attempt to compute the result of an exponentiation, the true value of which is a complex number with a non-zero imaginary part, an error condition occurs with ecode = "M95".

7.2.2 Truth operator truthop

truthop ::= | relation
logicalop |

7.2.2.1 Relational operator relation

relation ::= | = | == | < | <= | > | >= | [|] |]= |]] |]]= |

The operators =, ==, <, <=, >, >=, [,],]=,]], and]]= produce the truth value 1 if the relation between their operands which they express is true, and 0 otherwise. The dual operators '*relation* are defined by:

A '*relation* *B* has the same value as '(*A* *relation* *B*).

Editor's note:
The term "dual" was introduced into the standard in a day and age that all relational operators were single characters, and indicated that the described "combined" operators would have two characters. Suggest to replace the word "dual" with "multi-character".

7.2.2.2 Numeric relations

The inequalities <, <=, >, and >= operate on the numeric interpretations of their operands; they denote the conventional algebraic *less than*, *less than or equal to*, *greater than*, and *greater than or equal to*.

7.2.2.3 String relations

The relations =, [,],]=,]], and]]= do not imply any numeric interpretation of either of their operands.

The relation = tests string identity. If the operands are not known to be numeric and numeric equality is to be tested, the programmer may apply an appropriate unary operator to the non-numeric operands. If both operands are known to be in numeric form (as would be the case, for example, if they resulted from the application of any operator except _), application of a unary operator is not necessary. The uniqueness of the numeric representation guarantees the equivalence of string and numeric equality when both operands are numeric. Note, however, that the division operator / may produce inexact results, with the usual problems attendant to inexact arithmetic.

The relation [is called *contains*. *A* [*B* is true if and only if *B* is a substring of *A*; that is, *A* [*B* has the same value as `''$FIND(`*A*`,`*B*`)`. Note that the empty string is a substring of every string.

The relation] is called *follows*. *A*] *B* is true if and only if *A* follows *B* in the sequence, defined here. *A* follows *B* if and only if any of the following is true.

a. *B* is empty and *A* is not.

b. Neither *A* nor *B* is empty, and the leftmost character of *A* follows (i.e., has a numerically greater $ASCII value than) the leftmost character of *B*.

c. There exists a positive integer *n* such that *A* and *B* have identical heads of length *n*, (i.e., $EXTRACT(*A* , 1, *n*) = $EXTRACT(*B* , 1, *n*)) and the remainder of *A* follows the remainder of *B* (i.e., $EXTRACT(*A* , *n* +1, $LENGTH(*A*)) follows $EXTRACT(*B* , *n* +1, $LENGTH(*B*))).

The relation]= is called *follows or equals*. *A*]=*B* is true if and only if *A* follows *B* as defined above or *A* is identical to *B*.

The relation]] is called *sorts after*. *A*]]*B* is true if and only if *A* follows *B* in the subscript ordering sequence defined by the single argument $ORDER function as if that $ORDER refers to an lvn.

The relation]]= is called *sorts after or equals*. *A*]]=*B* is true if and only if *A* sorts after *B* as defined above or *A* is identical to *B*.

The relation == tests object reference identity. *A* == *B* is true if and only if all of the following conditions are met:

a. The value of *A* has the data type OREF.
b. The value of *B* has the data type OREF.
c. *A* and *B* identify the same object.

In the context of the == operator, values are never coerced to any specific data type or interpretation.

7.2.2.4 Logical operator logicalop

logicalop ::= | & | ! | !! |

The operators !, !!, and & are called logical operators. (They are given the names *or*, *exclusive or*, and *and*, respectively.) They operate on the truth-value interpretations of their operands, and they produce truth-value results.

A ! *B* = (0 if both *A* and *B* have the value 0)
(1 otherwise)

A !! *B* = (0 if both *A* and *B* have the value 0 or if both A and B have the value 1)
(1 otherwise)

A & *B* = (1 if both *A* and *B* have the value 1)
(0 otherwise)

The multi-character operators '&, '! And '!! are defined by:

A '& *B* = '(*A* & *B*)
A '! *B* = '(*A* ! *B*)
A '!! *B* = '(*A* !! *B*)

7.2.3 Pattern match pattern

The pattern match operator ? tests the form of the string which is its left-hand operand. *S* ? *P* is true if and only if *S* is a member of the class of strings specified by the pattern *P*.

A pattern is a concatenated list of pattern atoms.

pattern ::= | patatom ... | @ expratom V pattern |

Assume that pattern has *n* patatoms. *S* ? pattern is true if and only if there exists a partition of *S* into *n* substrings

$$S = S_1 S_2 ... S_n$$

such that there is a one-to-one order-preserving correspondence between the S_i and the pattern atoms, and each S_i *satisfies* its respective pattern atom. Note that some of the S_i may be empty.

Each pattern atom consists of a repeat count repcount, followed by either a pattern code patcode, an alternation or a string literal strlit. A substring S_i of *S* satisfies a pattern atom if it, in turn, can be decomposed into a number of concatenated substrings, each of which satisfies the associated patcode, alternation or strlit.

patatom ::= repcount | patcode
patstr
alternation | [patsetdest]

repcount ::= | intlit
[intlit$_1$] . [intlit$_2$] |

patcode ::= ['] | Y patnonY Y
Z patnonZ Z
patnonYZ
OB charspec CB | ...

patnonY ::= any of the characters in ident except Y

patnonZ ::= any of the characters in ident except Z

patnonYZ ::= any of the characters in ident except Y and Z

charspec ::= strconst$_1$ [: strconst$_2$]

strconst ::= | $C [HAR] (L numlit)
strlit |

patstr ::= ['] strlit

alternation ::= (L patgrp)

patgrp ::= patatom ...

patsetdest ::= (setdestination)

patcodes beginning with the initial letter Y are available for use by M[UMPS] programmers. patcodes beginning with the initial letter Z are available for use by implementors. patcodes are specified in Character Set Profiles.

a. If a patcode has the form of a charspec, determination of whether a character belongs to the patcode is made as follows: A character belongs to a charspec containing only one strconst if it is contained in the string represented by that strconst. A character belongs to a charspec containing two strconsts if it is (inclusively) between them. Formally, *X* is a member of *S* if *S* [*X*, and *X* is a member of *S1*:*S2* if *S1* does not trail *X* and *X* does not trail *S2*, and the check against the value of *S2* will be omitted if the value of *S2* is the empty string. If *S2* is present, then neither *S1* nor *S2* may contain more than one character.

   If a strconst is of the form $C[HAR](...), then it has the same value as the result of the function $CHAR called with the same parameters. Use of upper, lower, or mixed case in the name $CHAR is permitted

b. Otherwise, patcodes differing only in the use of corresponding upper and lower case letters are equivalent. If the apostrophe is not present in a given patcode, the patcode is satisfied by any single character in the union of the classes of characters represented, each class denoted by its own patcode letter. If the apostrophe is present, the patcode is satisfied by any single character which is not in the union of the classes of characters represented. Whether or not a specific character belongs to a patcode class is determined by a process' Character Set Profile (charset).

An alternation is satisfied if any one of its patgrp components individually matches the corresponding S_i.

Each patstr in which an apostrophe is not present is satisfied by, and only by, the value of strlit. Each patstr in which an apostrophe is present is satisfied by any string of the same length as strlit which is not identical to strlit.

If repcount has the form of an indefinite multiplier ".", patatom is satisfied by a concatenation of any number of S_i (including none), each of which meets the specification of patatom.

If repcount has the form of a single intlit, patatom is satisfied by a concatenation of exactly intlit instances of S_i, each of which meets the specification of patatom. In particular, if the value of intlit is zero, the corresponding S_i is empty.

If repcount has the form of a range, $intlit_1$.$intlit_2$, $intlit_1$ gives the lower bound, and $intlit_2$ the upper bound. If the upper bound is less than the lower bound an error condition occurs with ecode = "M10". If the lower bound is omitted, so that the range has the form .$intlit_2$, the lower bound is taken to be zero. If the upper bound is omitted, so that the range has the form $intlit_1$. , the upper bound is taken to be indefinite; that is, the range is at least $intlit_1$ occurrences. Then patatom is satisfied by the concatenation of a number of S_i, each of which meets the specification of patatom, where the number must be within the expressed or implied bounds of the specified range, inclusive.

If more than one one-to-one order-preserving correspondence between the S_i and the pattern atoms exist the following rules are used to select the correspondence used in the two paragraphs following the rules. These rules are applied to each patatom in the pattern, from left to right and recursively in the case of alternations.

a. If the patatom is not an alternation, select the longest matching substring that produces a match in the pattern as a whole.

b. If the patatom is an alternation, use the below rules and apply rules A and B recursively to each patatom in the selected patgrp(s) from left to right.

   1. Select the correspondence(s) that use(s) the smallest possible value of the alternation's repcount.

   2. If multiple correspondences satisfy 1), for each sequential application of the alternation (i.e., each value of the repcount) select the patgrp(s) within the alternation that correspond to the longest possible substring.

3. If multiple correspondences satisfy 1) and 2), select the leftmost patgrp in the alternation.

Each optional patsetdest, if any, is executed only if S?pattern is true, and only if the associated pattern atom is satisfied by one of the S_i in the selected correspondence. If these conditions hold, these (and only these) patsetdests are executed from left to right as follows:

> For each of the substrings S_i of S satisfying the pattern atom in the selected correspondence, in the order in which they (the S_i) appear in the string, perform all the actions of SET setdestination=S_i as defined in section 8.2.30.

The multi-character operator '? is defined by:

> *A* '? *B* = '(*A* ? *B*)

8 Commands

8.1 General command rules

Every command starts with a commandword which dictates the syntax and interpretation of that command instance. commandwords differing only in the use of corresponding upper and lower case letters are equivalent. The standard contains the following commandwords:

commandword ::=

AB[LOCK]
A[SSIGN]
ASTA[RT]
ASTO[P]
AUNB[LOCK]
B[REAK]
C[LOSE]
D[O]
E[LSE]
ESTA[RT]
ESTO[P]
ETR[IGGER]
F[OR]
G[OTO]
H[ALT]
H[ANG]
I[F]
J[OB]
K[ILL]
KV[ALUE]
KS[UBSCRIPTS]
L[OCK]
M[ERGE]
N[EW]
O[PEN]
Q[UIT]
R[EAD]
RL[OAD]
S[ET]
TC[OMMIT]
TRE[START]
TRO[LLBACK]
TS[TART]
U[SE]
V[IEW]
W[RITE]
X[ECUTE]
Z[unspecified]

Unused commandwords other than those starting with the letter "Z" are reserved for future extensions to the standard.

Any implementation of the language must be able to recognize both the abbreviated commandword (i.e., the character(s) to the left of the "[" in the list above) and the full spelling of each commandword. When two commands have a common abbreviated commandword, their argument syntax uniquely distinguishes them.

The formal definition of the syntax of command is a choice from among all of the individual command syntax definitions of 8.2.

command	::=	syntax of ASSIGN command syntax of BREAK command ... syntax of XECUTE command syntax of Z[unspecified] command

For all commands allowing multiple arguments, the form

commandword arg_1, arg_2, ... arg_n

is equivalent in execution to

commandword arg_1 commandword arg_2 ... commandword arg_n

Within a command, all expratoms are evaluated in a left-to-right order with all expratoms that occur to the left of the expratom being evaluated, including the complete resolution of any indirection, prior to the evaluation of that expratom, except as explicitly noted elsewhere in this document. The expratom is formed by the longest sequence of characters that satisfies the definition of expratom. (See 7.1 for a description of expratom).

An error condition occurs, with ecode = "M11", when execution begins of any formalline unless that formalline has just been reached as a result of an exvar, an exfunc, a JOB command jobargument, or a DO command doargument that contains an actuallist.

8.1.1 Spaces in commands

Spaces are significant characters. The following rules apply to their use in lines.

a. If a command instance contains no argument and it is not the last command of the line, or if a comment or extsyntax follows, the commandword or postcond is followed by at least two spaces. If it is the last command of the line and no comment or extsyntax follows, the commandword or postcond may be followed by zero or more spaces.

b. In all other cases, the use of spaces is defined by the appropriate command definition and subsections 6.2 Routine body, and 6.4 Embedded programs.

8.1.2 Comment comment

If a semicolon appears in the commandword initial-letter position, it is the start of a comment. The remainder of the line to eol must consist of graphics only, but is otherwise ignored and non-functional.

8.1.3 Command argument indirection

Indirection is available for evaluation of either individual command arguments or contiguous sublists of command arguments. The opportunities for indirection are shown in the syntax definitions accompanying the command descriptions.

Typically, where a commandword carries an argument list, as in

commandword SP L argument

the argument syntax will be expressed as

argument	::=	individual argument syntax @ expratom V L argument

This formulation expresses the following properties of argument indirection.

a. Argument indirection may be used recursively.

b. A single instance of argument indirection may evaluate to one complete argument or to a sublist of complete arguments.

Unless the opposite is explicitly stated, the text of each command specification describes the arguments *after* all indirection has been evaluated.

Unless expressed otherwise, if individual argument syntax allows the @ expratom contruct, then argument indirection has precedence, i.e., the restriction on the value of expratom comes from the V operator of the argument indirection, not any other type of indirection.

8.1.4 Post conditional postcond

All commands except ELSE, FOR, and IF may be made conditional as a whole by following the commandword immediately by the post-conditional postcond.

postcond ::= [: tvexpr]

If the postcond is absent or the postcond is present and the value of the tvexpr is true, the command is executed. If the postcond is present and the value of the tvexpr is false, the commandword and its arguments are passed over without execution.

The postcond may also be used to conditionalize the arguments of DO, GOTO, and XECUTE. In such cases the arguments' expratoms that occur prior to the postcond are evaluated prior to the evaluation of the postcond.

8.1.5 Command timeout timeout

The OPEN, LOCK, JOB, and READ commands employ an optional timeout specification, associated with the testing of an external condition.

timeout ::= : numexpr

If the optional timeout is absent, the command will proceed if the condition, associated with the definition of the command, is satisfied; otherwise, it will wait until the condition is satisfied and then proceed.

$TEST will not be altered if the timeout is absent.

If the optional timeout is present, the value of numexpr must be non-negative. If it is negative, the value 0 is used. Numexpr denotes a *t*-second timeout, where *t* is the value of numexpr.

> If *t* = 0, the condition is tested. If it is true, $TEST is set to 1; otherwise, $TEST is set to 0. Execution proceeds without delay.
>
> If *t* is positive, execution is suspended until the condition is true, but in any case no longer than *t* seconds. If, at the time of resumption of execution, the condition is true, $TEST is set to 1; otherwise, $TEST is set to 0.

8.1.6 Line reference lineref

The DO , GOTO, and JOB commands, extrinsic functions and extrinsic variables, as well as the $TEXT function, contain in their arguments means for referring to particular lines within any routine. This subclause describes the means for making line references.

A reference to a line is either an entryref or a labelref. An entryref allows the specification of integer offsets from a label (eg, LOOP+5 references the fifth line after the line that has LOOP for a label). Also, an entryref allows indirection of both the label and the routinename. A labelref, on the other hand, allows neither label offsets nor indirection.

lineref ::= | entryref
labelref |

8.1.6.1 Entry reference entryref

The total line specification in DO, GOTO, JOB, and $TEXT is in the form of entryref.

entryref ::= | dlabel [+ intexpr] [^ routineref]
^ routineref |

If the routine reference (^ routineref) is absent, the routine being executed is implied. If the line reference (dlabel [+intexpr]) is absent, the first line is implied.

If +intexpr is absent, the line denoted by dlabel is the one containing label in a defining occurrence. If +intexpr is present and has the value *n* ‘< 0, the line denoted is the *n*th line after the one containing label in a defining occurrence. A negative value of intexpr causes an error condition with ecode = "M12". When label is an instance of intlit, leading zeros are significant to its spelling.

In the context of DO , GOTO, or JOB, either of the following conditions causes an error condition with ecode = "M13".

a. A value of intexpr so large as not to denote a line within the bounds of the given routine.

b. A spelling of label which does not occur in a defining occurrence in the given routine.

In any context, reference to a particular spelling of label which occurs more than once in a defining occurrence in the given routine will have undefined results.

Editor’s note:
Section 6.2.3 specifies that such a label cannot exist, and that an error with ecode = M57 occurs at an attempt to create such a label.

Recommendation in X11/1999-7: change this sentence to:
In any context, reference to a particular spelling of label which occurs more than once in a defining occurrence in the given routine will not be possible, because the insertion of such a duplicate label will cause an error with ecode = “M57".

DO, GOTO, and JOB commands, as well as the $TEXT function, can refer to a line in a routine other than that in which they occur; this requires a means of specifying a routinename.

Any line in a given routine may be denoted by mention of a label which occurs in a defining occurrence on or prior to the line in question.

```
dlabel ::= | label                          |
           | @ expratom V dlabel            |

routineref ::= | [ VB environment VB ] routinename |
               | @ expratom V routineref          |
```

If the routineref includes an environment, then the routine is fetched from the specified environment. Reference to a non-existent environment causes an error condition with an ecode = "M26".

8.1.6.2 Label reference labelref

When the DO or JOB commands or exfunc include parameters to be passed to the specified routine, the +intexpr form of entryref is not permitted and the specified line must be a formalline. The line specification labelref is used instead:

```
labelref ::= | label [ ^ [ VB environment VB ] routinename ] |
             | ^ [ VB environment VB ] routinename            |
```

If the routine reference (^ [VB environment VB] routinename) is absent, the routine being executed is implied. If the line reference (label) is absent, the first line is implied. If the labelref includes an environment, then the routine is fetched from the specified environment. Reference to a non-existent environment causes an error condition with an ecode = "M26".

In the context of a DO or JOB command, an exfunc, or an exvar, a spelling of label which does not occur in a defining occurrence in the given routine causes an error condition with ecode = "M13".

8.1.6.3 External reference externref

```
externref ::= & [ packagename . ] externalroutinename

packagename ::= name

externalroutinename ::= name [ ^ name ]
```

The ampersand (&) character designates a program whose namespace is external to the current M environment. The effects of passing parameters are as defined in 8.1.7 (Parameter Passing).

The packagename shall be from a namespace of those determined by the appropriate namespace registry. If packagename is not specified, implementors may, optionally, choose to provide a default package.

Bindings may have one or more namespaces; requirements to use these namespaces must be clearly stated in the specification of the binding. The term *package* is used herein to denote programs that are in possibly external environments. No implied one-to-one correspondence for all possible external packages exists.

The externalroutinename namespace is not defined in this document; this is a function of a binding. Any external mapping between the externalroutinename and any name used by an external package is an implementation-specific issue. The externalroutinename shall be of the form name or $name_1$^$name_2$.

8.1.6.4 Library reference libraryref

```
libraryref ::= % libraryelement [ ^ library ]

libraryelement ::= name
```

library ::= name

If no library is specified as part of a libraryref then the libraries specified in ^$JOB($JOB , "LIBRARY") are used. Note: This does not imply that the libraries specified in ^$JOB($JOB , "LIBRARY") can necessarily be dynamically changed during the lifetime of a process.

Unless explicitly specified in an individual libraryelement definition accessing a libraryref has no effect on local variables for a process, $REFERENCE, and $TEST, except for a return value and changes to variables passed by reference.

If an argument to a libraryref has an invalid value (such as a value outside the domain of the function) the behavior of the reference to the libraryref is undefined.

The restrictions specified in 8.1.7 Parameter passing also apply to the referencing of librarysrefs.

If a libraryelement or a library is not available for a library reference then an error condition occurs with ecode = "M13".

8.1.7 Parameter passing

Parameter passing is a method of passing information in a controlled manner to and from a subroutine or process as the result of an exfunc, or a DO command with an actuallist, or to a process as the result of a JOB command with an actuallist.

actuallist ::= ([L actual])

actual ::= [. actualname | expr]

actualname ::= | name | @ expratom V actualname |

When parameter passing occurs, the formalline designated by the labelref must contain a formallist in which the number of names is greater than or equal to the number of actuals in the actuallist. The correspondence between actual and formallist name is defined such that the first actual in the actuallist corresponds to the first name in the formallist, the second actual corresponds to the second formallist name, et cetera. Similarly, the correspondence between the parameter list entries, as defined below, and the actual or formallist names is also by position in left-to-right order. If the syntax of actual is .actualname, then it is said that the actual is of the call-by-reference format; if the syntax of actual is expr it is said that the actual is of the call-by-value format; otherwise it is said that the actual is of the omitted-parameter format.

When parameter passing occurs, the following steps are executed:

a. Process the actuals in left-to-right order to obtain a list of DATA-CELL pointers called the parameter list. The parameter list contains one item per actual. The parameter list is created according to the following rules:

    1. If the actual is call-by-value, then evaluate the expr and create a DATA-CELL with a zero tuple value equal to the result of the evaluation. An expr that returns a value of data type OREF is coerced into a value of data type MVAL in the actuallist of externrefs and JOB commands (See 7.1.1.1.2 for the coercion rules), but not in any other actuallists. The pointer to this DATA-CELL is the parameter list item.

    2. If the actual is call-by-reference, search the NAME-TABLE for an entry containing the actual name. If an entry is found, the parameter list item is the DATA-CELL pointer in this NAME-TABLE entry. If the actual name is not found, create a NAME-TABLE entry containing the

name and a pointer to a new (empty) DATA-CELL. This pointer is the parameter list item. If a jobargument contains a call-by-reference actual an error occurs with ecode = "M40" .

3. If the actual is an omitted-parameter, create a new (empty) DATA-CELL.

b. Place the information contained in the formallist in the PROCESS-STACK frame.

c. For each name in the formallist, search the NAME-TABLE for an entry containing the name and if the entry exists, copy the NAME-TABLE entry into the parameter frame and delete it from the NAME-TABLE. This step performs an implicit NEW on the formallist names.

d. For each item in the parameter list, create a NAME-TABLE entry containing the corresponding formallist name and the parameter list item (DATA-CELL pointer). This step binds the formallist names to their respective actuals.

As a result of these steps, two (or more) NAME-TABLE entries may point to the same DATA-CELL. As long as this common linkage is in effect, an ASSIGN, SET, or KILL of an lvn with one of the names appears to perform an implicit ASSIGN, SET, or KILL of an lvn with the other name(s). Note that a KILL does not undo this linkage of multiple names to the same DATA-CELL, although subsequent parameter passing or NEW commands may.

Execution is then initiated at the first command following the ls of the line specified by the labelref. Execution of the subroutine continues until an eor or a QUIT is executed that is not within the scope of a subsequently executed doargument, argumentless DO, xargument, exfunc, exvar, or FOR. In the case of an exfunc or exvar, the subroutine must be terminated by a QUIT with an argument.

At the time of the QUIT, the formallist names are unbound and the original saved values, including any undefined states, of the variables named in the formallist are restored. See 8.2.26 for a discussion of the semantics of the QUIT operation.

When calling to an externref, call-by-reference has the following additional implementation independent definition:

a. Upon return of control to M[UMPS], changes to the value of the lvn referenced by the actualname shall be as if the lvn was modified by an ASSIGN or SET command. The exact mechanism performing this operation is unspecified.

b. The resultant events are unspecified, if the data in the M[UMPS] environment is modified while an external routine call is being made that references the modified data.

c. Local variables (see 7.1.1.2 Variables) that are not passed as parameters, will not necessarily be available to the external environment.

8.1.8 Object usage

An *object* is an identifiable, encapsulated entity that has state and that provides one or more services, called *methods* and *properties*. A *service* may be accessed from a routine. The only means of observing or changing the state or behavior of an object is by use of its service(s).

Objects are not named. A value of data type OREF (object reference) is a value that identifies an object in an implementation-specific way. A value may be of one of two data types: either data type OREF or data type MVAL.

A value of data type OREF may be assigned to an lvn and may be used only in certain specified contexts. (See 7.1.1.1.2 Values of data type OREF)

8.1.8.1 Accessing a service

A *service* is identified by a name, called a servicename.

servicename ::= | name
strlit |

A *service* is accessed explicitly by means of an owservice (object with service):

owservice ::= | owmethod
owproperty |

owmethod ::= object . fservice

owproperty ::= object . [fservice]

object ::= expratom V oref

fservice ::= servicename [namedactuallist]

namedactuallist ::= ([L actual [, L namedactual]
L namedactual])

namedactual ::= actualkeyword := actual

actualkeyword ::= | name
strlit |

The object specifies an *object*, and the fservice specifies the *service* to be requested of that *object*.

If an expratom is used in a context where an object is expected, and the expratom does not return a value of the data type OREF, an error condition will occur with ecode = "M108" (Not an Object).

If, in the context of an owmethod or an owproperty, an expratom returns a value of data type OREF that refers to an *object* that is not currently accessible, an error condition will occur with ecode = "M105" (Inaccessible Object).

If an fservice fails to specify a *service* provided by the *object*, or that *service* does not support the context of the access, an error condition will occur with ecode = "M106" (Invalid Service). In the case of a *property*, if no fservice is specified, the value of the default *property*, if any, for the *object* is used. If there is no default *property*, an error condition will occur with ecode = "M107" (No Default Value).

A namedactuallist may contain positional parameters and named parameters. An actualkeyword specifies the name of the parameter in the *service* being accessed that will receive the actual. An actual without an actualkeyword is a positional parameter.

Note: names of *services* and parameters that do not conform to the syntax of a name can be used with external *objects*. In these cases, the name of the *service* or parameter must be represented as a strlit, i.e., such names must be enclosed in quotation marks, and any quotation marks within the name must be spelled twice. A strlit that evaluates to a name is equivalent to that name when used as a servicename or an actualkeyword.

Upon completion of a *service*, the values of $TEST and the naked indicator will be restored to their respective values prior to execution of the *service*.

The meaning of invoking an *object*'s *services*, either implicitly or explicitly, when the value of $TLEVEL is greater than 0 is reserved.

8.1.9 User-defined mnemonicspaces

When a controlmnemonic is used for a device which has a user-defined mnemonicspace (see 8.2.25) then the usage of the controlmnemonic in a READ and WRITE command format in the form

```
/controlmnemonic(expr,...)
```

is computationally equivalent, with the exception of the effect on $TEST and the naked indicator, to

```
DO label^routine(expr,...)
```

where *routine* is the user-defined mnemonicspace routine and *label* is controlmnemonic, unless controlmnemonic commences with a ? in which case it is replaced by %. If the controlmnemonics of the mnemonicspace are case-insensitive then *label* is controlmnemonic converted to upper-case. Unless specifically stated otherwise mnemonicspaces are case-sensitive.

$TEST and the naked indicator are restored to their value prior to the execution of the controlmnemonic associated routine. $TEST is not restored if there is a timeout on the original command.

Any reference to a controlmnemonic within a user-defined mnemonicspace for which there is no associated line causes an error condition with ecode = "M32".

devicecommand ::= | CLOSE
OPEN
USE |

If a label of the form %*command*, where *command* is a devicecommand, exists in a mnemonicspace command routine then execution of a command which is a devicecommand with at least one deviceparam is computationally equivalent to

```
NEW KEYWORD,ATTRIB,I
SET (KEYWORD,ATTRIB)=no
FOR I=1:1:no DO
. SET KEYWORD(I)=key_I
. IF $DATA(att_I) SET ATTRIB(I)=att_I
. QUIT
DO %label^routine(expr,.KEYWORD,.ATTRIB,time)
```

where *label* is the commandword converted to upper-case and expanded to the fully spelled out devicecommand, *routine* is the user-defined mnemonicspace command routine, *no* is the number of deviceparams, KEYWORD and ATTRIB contain the individual deviceparams in deviceparameters fully evaluated with key_i=devicekeyword$_i$ or deviceattribute$_i$ as appropriate and att_i=expr$_i$ if deviceparam is in the deviceattribute form, and *time* is absent or the evaluated expression from timeout if timeout is present.

The usage of the deviceparam form expr is implementation specific.

Any action implied by the presence of a mnemonicspace in such a command takes effect before the above code is executed.

iocommand ::= | READ
WRITE |

If a label of the form *%command,* where *command* is an iocommand, exists in a mnemonicspace

command routine then execution of an iocommand of the form:

a. W[RITE] ffformat
b. W[RITE] nlformat
c. W[RITE] tabformat
d. W[RITE] expr
e. W[RITE] * intexpr
f. R[EAD] glvn [readcount] [timeout]
g. R[EAD] ffformat
h. R[EAD] nlformat
i. R[EAD] tabformat
j. R[EAD] strlit
k. R[EAD] * glvn [timeout]

is respectively computationally equivalent, with the exception of the effect on $TEST and the naked indicator, to:

a. DO %WRITEFF^*routine*()
b. DO %WRITENL^*routine*()
c. DO %WRITETAB^*routine*(intexpr)
d. DO %WRITE^*routine*(expr)
e. DO %WRITES^*routine*(intexpr)
f. SET glvn=$$%READ^*routine*($intexpr_1$ [, $intexpr_2$])
g. DO %WRITEFF^*routine*(1)
h. DO %WRITENL^*routine*(1)
i. DO %WRITETAB^*routine*(intexpr ,1)
j. DO %WRITE^*routine*(strlit ,1)
k. SET glvn=$$%READS^*routine*([, $intexpr_2$])

where:
1. *routine* is the user-defined mnemonicspace command routine,
2. $intexpr_1$ is the intexpr from readcount, or absent if no readcount is present, and
3. if timeout is present, $intexpr_2$ is the intexpr from timeout.

It is the responsibility of the user-defined mnemonicpace routine to process the deviceparameters in the appropriate order.

During the execution of any user-defined mnemonicspace command routine:

a. READ and WRITE redirection for the device that caused the routine to be executed is disabled; and

b. the effect of user-defined processing of controlmnemonics and commands for the same mnemonicspace is unspecified.

Upon completion of execution of the user-defined mnemonicspace routine, the naked indicator is restored to its original value.

In the event that a timeout is *not* present, $TEST is also restored when execution of the routine completes. However, if a timeout is present, $TEST is *not* restored and it is the responsibility of the user-defined mnemonicspace routine to return $TEST to indicate whether the operation times out.

Note: $STORAGE may be affected by the execution of user-defined mnemonicspace code.

8.2 Command definitions

The specifications of all commands follow.

8.2.1 ABLOCK

AB[LOCK] postcond | [SP] SP ablockargument |

ablockargument ::= | L evclass (L evclass) |

evclass ::= expr V " | COMM IPC INTERRUPT POWER TIMER USER Z[unspecified] | "

Event classes not specified above are reserved for future extensions to the standard.

ABLOCK temporarily blocks events during critical sections of a process. The three forms of ABLOCK are given the following names:

a) L evclass Selective ABLOCK
b) (L evclass) Exclusive ABLOCK
c) Empty argument list: ABLOCK All

In the Selective ABLOCK, the named event classes are blocked as described below. In the Exclusive ABLOCK, all event classes except the named event classes are blocked as described below. In the ABLOCK All, all event classes are blocked as described below.

When an event class is blocked, an internal counter for that event class is incremented. If the counter has a positive value, all events of that class are blocked from interrupting the process executing the ABLOCK command. If a registered event occurs while blocked, the event is queued. Unregistered events are not queued. Additional subsequent events may be queued if space is provided by the implementation (space for only one event is guaranteed). Events, if queued, will occur in the order in which they occurred when the block is removed (i.e., when the counter becomes zero). All events for a process are stored in one of two queues (one for synchronous events, the other for asynchronous events), rather than a separate queue for each class. Each process, however, must maintain its own queues, as each process blocks and unblocks events independently.

8.2.2 ASSIGN

A[SSIGN] postcond SP L assignargument

assignargument ::= | assigndestination = object @ expratom V L assignargument |

assigndestination ::= | assignleft (L assignleft) |

assignleft ::= | lvn owproperty |

ASSIGN is a special means for explicitly assigning a reference to an *object* to an lvn. The ASSIGN command behaves similar to the SET command, with the exception that the final value of the expr to the right-hand side of the = sign must be of data type OREF and will not be coerced into a value of data type MVAL. (See notes under 7.1.1.1.2 for the results of the ASSIGN command with various operands)

This special behavior allows the ASSIGN command to transfer the value of data type OREF to the assignleft.

8.2.3 ASTART

ASTA[RT] postcond | [SP] SP ablockargument |

ASTART enables asynchronous event processing for all or selected event classes. The three forms of ASTART are given the following names:

a) L evclass	Selective ASTART
b) (L evclass)	Exclusive ASTART
c) Empty argument list:	ASTART All

In the Selective ASTART, the named event classes are enabled for asynchronous event processing as described below. In the Exclusive ASTART, all event classes except the named event classes are enabled for asynchronous event processing as described below. In the ASTART All, all event classes are enabled for asynchronous event processing as described below.

If any of the classes being enabled for asynchronous event processing are currently enabled for synchronous event processing an error occurs with an ecode = "M102".

Event classes are enabled by ASTART only for the process executing the ASTART command. It is not an error to enable an event class which is already enabled for the asynchronous model.

8.2.4 ASTOP

ASTO[P] postcond | [SP] SP ablockargument |

ASTOP disables asynchronous event processing for all or selected event classes. The three forms of ASTOP are given the following names:

a) L evclass	Selective ASTOP
b) (L evclass)	Exclusive ASTOP
c) Empty argument list:	ASTOP All

In the Selective ASTOP, the named event classes are disabled for asynchronous event processing as described below. In the Exclusive ASTOP, all event classes except the named event classes are disabled for asynchronous event processing as described below. In the ASTOP All, all event classes are disabled for asynchronous event processing as described below.

When asynchronous event processing is disabled for a given event class, events of that class have no effect on the process. Event classes are disabled by ASTOP only for the process executing the ASTOP command. It is not an error to disable an event class which is already disabled.

8.2.5 AUNBLOCK

AUNB[LOCK] postcond | [SP] SP ablockargument |

AUNBLOCK removes a temporary block on events that was imparted by ABLOCK. The three forms of AUNBLOCK are given the following names:

Editor's note:
"imparted" doesn't seem to be thre right word. Suggest to change it to "created".

a) L evclass Selective AUNBLOCK
b) (L evclass) Exclusive AUNBLOCK
c) Empty argument list: AUNBLOCK All

In the Selective AUNBLOCK, the named event classes are unblocked as described below. In the Exclusive AUNBLOCK, all event classes except the named event classes are unblocked as described below. In the AUNBLOCK All, all event classes are unblocked as described below.

When an event class is unblocked, the internal counter for the event class (see page 94, ABLOCK) is decremented, unless it is already zero (the counter may not be negative). If the counter is zero, the temporary block, if any, on the event class is removed. Pending events (see page 94, ABLOCK), if any, occur in the order in which they arrived. Blocks are removed only for the process executing the AUNBLOCK command. It is not an error to unblock events which are not currently blocked.

8.2.6 BREAK

B[REAK] postcond | [SP] argument syntax unspecified |

BREAK provides an access point within the standard for non-standard programming aids. BREAK without arguments suspends execution until receipt of a signal, not specified here, from a device.

8.2.7 CLOSE

C[LOSE] postcond SP L closeargument

closeargument ::= | devn [: deviceparameters] @ expratom V L closeargument |

devn ::= [VB environment VB] expr

deviceparameters ::= | deviceparam ([[deviceparam] :] ... deviceparam) |

deviceparam ::= | expr devicekeyword deviceattribute = expr |

devicekeyword ::= [/] name

deviceattribute ::= [/] name

The order of execution of deviceparams is from left to right within a deviceparameters usage.

If there is no mnemonicspace in use for a device or the current mnemonicspace is the empty string then the implementation may allow any of the forms of deviceparam. The expr form may not be mixed with the other forms within the same deviceparameters.

In all other cases the expr form is not allowed.

devn identifies a device. (In this clause, the term *device* encompasses I/O devices, files, data sets, and other objects supporting OPEN, USE, READ, WRITE, and CLOSE commands.) When environment is omitted, the value of expr denotes one device. When environment is present, the value of environment denotes one set of devices, while the value of expr denotes one member of the set. The interpretation of the values is left to the implementor. Reference to a non-existent environment causes an error condition with an ecode = "M26".

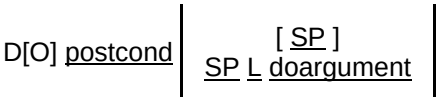
The deviceparameters may be used to specify termination procedures or other information associated with relinquishing ownership, in accordance with implementor interpretation.

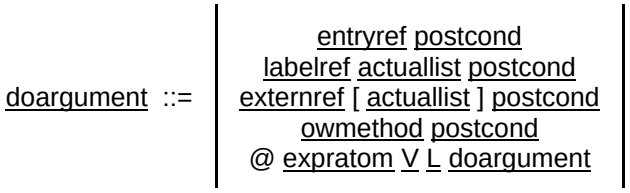
When a deviceparam is encountered that contains a devicekeyword for which there is no defined meaning in the current mnemonicspace, the implementation may or may not cause an error to happen. If an error occurs, the ecode will contain “,M109,”.

When a deviceparam is encountered that contains a deviceattribute for which there is no defined meaning in the current mnemonicspace, the implementation may or may not cause an error to happen. If an error occurs, the ecode will contain “,M109,”.

Each designated device is released from ownership. If a device is not owned at the time that it is named in an argument of an executed CLOSE, the command has no effect upon the ownership and the values of the associated parameters of that device. Device parameters in effect at the time of the execution of CLOSE are retained for possible future use in connection with the device to which they apply. (See 8.3.1, which specifies an exception for output time-out.) If the current device is named in an argument of an executed CLOSE command, $IO is given a value of the empty string.

8.2.8 DO





An argumentless DO initiates execution of an inner block of lines. If postcond is present and its tvexpr is false, the execution of the command is complete. Otherwise, i.e. if postcond is absent, or the postcond is present and its tvexpr is true, the DO places a DO frame on the PROCESS-STACK containing the current execution location, the current execution level, and the current value of $TEST, increases the execution level by one, and continues execution at the next line in the routine. (See 6.3 for an explanation of routine execution.) When encountering an implicit or explicit QUIT not within the scope of a subsequently executed doargument, argumentless DO, xargument, exfunc, exvar, or FOR, execution of this block is terminated (see 8.2.26 for a description of the actions of QUIT). Execution resumes at the command (if any) following the argumentless DO.

DO with arguments is a generalized call to the subroutine specified by the entryref, the labelref, the externref, or to the method specified by the owmethod in each doargument. The line specified by the

entryref or labelref, must have a LEVEL of one. If the line specified is an externref then an implicit LEVEL of 1 is assumed, unless otherwise specified within the binding. Execution of a doargument to a line whose LEVEL is not one causes an error condition with ecode = "M14".

If the actuallist is present in an executed doargument, parameter passing occurs and the formalline designated by labelref must contain a formallist in which the number of names is greater than or equal to the number of actuals in the actuallist. If the call is to an externref and an actuallist is present, then parameter passing occurs, and data is transferred (with any conversion as defined in the binding to the external package).

Each doargument is executed, one at a time in left-to-right order, in the following steps.

a. Evaluate the expratoms of the doargument.

b. If postcond is present and its tvexpr is false, execution of the doargument is complete. Otherwise, i.e. if postcond is absent, or postcond is present and its tvexpr is true, proceed to the step c.

c. A DO-frame containing the current execution location and the execution level are placed on the PROCESS-STACK.
   Note that the value of $TEST is not stacked in this case.

d. If the actuallist is present, execute the sequence of steps described in 8.1.7 Parameter Passing.

e. Continue execution at the first command position specified by the reference as follows:

   1. For entryref and labelref, this is the first command that follows the ls of the line specified by entryref or labelref. Execution of the subroutine (within the M[UMPS] environment) continues until an eor or a QUIT is executed that is not within the scope of a subsequently executed FOR, argumentless DO, doargument, xargument, exfunc, or exvar. The scope of this internally referenced doargument is said to extend to the execution of that QUIT or eor. (See 8.2.26 for a description of the actions of QUIT.) Execution then returns to the first character position following the doargument.

   2. For externref, this is the first executable item as specified within the package environment. If the reference is external to M[UMPS], execution proceeds in the specified environment until termination, as defined within that environment, occurs. Execution then returns to the first character following the doargument.

   3. For owmethod, refer to clause 8.1.8 Object usage.

8.2.9 ELSE

E[LSE] [SP]

If the value of $TEST is 1, the remainder of the line to the right of the ELSE command is not executed. If the value of $TEST is 0, execution continues normally at the next command.

8.2.10 ESTART

ESTA[RT] postcond | [SP] SP estartargument |

estartargument ::= | L wevclass (L wevclass) |

wevclass ::= | evclass
expr V "WAPI" |

ESTART enables synchronous event processing for the selected event classes. The additional class "WAPI" is provided to enable just the synchronous event processing specified in X11.6, the MWAPI. If any of the event classes being enabled for synchronous event processing is currently enabled for asynchronous event processing, an error occurs with ecode = "M102". It is not an error to enable an event class which is already enabled for synchronous event processing.

Synchronous event processing remains activated until the termination of execution of the ESTART command, except that synchronous event processing is implicitly deactivated at the initiation of call back processing for each event. At the conclusion of call back processing for each event, synchronous event processing is implicitly reactivated.

The tree forms of ESTART are given the following names:

a) L evclass	Selective ESTART
b) (L evclass)	Exclusive ESTART
c) Empty argument list:	ESTART All

In the Selective ESTART, the named event classes are enabled for synchronous event processing as described below. In the Exclusive ESTART, all event classes except the named event classes are enabled for synchronous event processing as described below. In the ESTART All, all event classes are enabled for synchronous event processing as described below.

When synchronous event processing is enabled for a given event class, events of that class will cause the execution of the registered event handler, if any, for that specific event (call back processing). Event classes are enabled by ESTART only for the process executing the ESTART command.

Call back processing can execute an ESTART command. In this case, the effect is to change the event classes which are enabled for subsequent synchronous event processing. ESTART commands are not nested. It is not an error to issue multiple ESTART commands on the same event class.

The execution of an ESTART command which starts synchronous event processing is terminated when an ESTOP command is executed during call back processing for that ESTART command. When execution of an ESTART command which starts synchronous event processing is terminated, execution continues with the command following that ESTART command.

8.2.11 ESTOP

ESTO[P] postcond [SP]

The ESTOP command implicitly performs the number of QUIT commands necessary to return to the execution level of the most recently executed ESTART command that started synchronous event processing, and then terminates that ESTART command. If synchronous event processing is not activated, execution of an ESTOP command has no effect. It is not possible to ESTOP only selected event classes.

8.2.12 ETRIGGER

ETR[IGGER] postcond SP especref

especref ::= | expr V ^$W[INDOW] (espec) [: einforef]
expr V ^$J[OB] (erspec) |

einforef ::=	See X11.6		[1]
espec ::=	See X11.6		[2]

erspec ::= processid , $expr_1$ V "EVENT", $expr_2$ V evclass, $expr_3$ V evid

evid ::= expr

Note that the definitions of espec and einforef are copied from X11.6, the MWAPI. Note also that the range of values allowed for evid depends on the value of evclass, and may be implementation-specific.

ETRIGGER causes an event to occur, though use of a processid other than the current job's own processid may be restricted by the implementation. This restricted use does not generate an error, but will not generate an event. Restrictions (if any) must be specified in the implementation's conformance statement.

If the use is not restricted and the specified event is enabled for either synchronous or asynchronous event processing, the event processing for it will occur subsequently. The event that occurs is specified by evclass and evid. If evid does not specify a valid event, an error condition occurs with an ecode = "M103".

If evclass evaluates to "IPC" and evid is not the current job's processid an error condition occurs with an ecode = "M104".

8.2.13 FOR

F[OR] | [SP] SP lvn = L forparameter |

forparameter ::=	expr $numexpr_1$: $numexpr_2$: $numexpr_3$ $numexpr_1$: $numexpr_2$	

The *scope* of the FOR command begins at the next command following the FOR on the same line and ends just prior to the eol on this line.

The FOR command with an argument specifies repeated execution of the commands within its scope for different values of the local variable lvn, under successive control of the forparameters, from left to right.

---

[1]

Editor's note:
The definition in X11.6 reads:

einforef ::=	glvn @ expratom V einforef	

[2]

Editor's note:
The definition in X11.6 reads:

espec ::= L expr

Any expressions occurring in lvn, such as might occur in subscripts or indirection, are evaluated once per execution of the FOR command, prior to the first execution of any forparameter.

For each forparameter, control of the execution of the commands in the scope is specified as follows. (Note that *A*, *B*, and *C* are hidden temporary variables.)

a. If the forparameter is of the form $expr_1$.

    1. Set lvn = expr.
    2. Execute the commands in the scope of the FOR command once.
    3. Processing of this forparameter is complete.

b. If the forparameter is of the form $numexpr_1$: $numexpr_2$: $numexpr_3$ and $numexpr_2$ is non-negative.

    1. Set *A* = $numexpr_1$.
    2. Set *B* = $numexpr_2$.
    3. Set *C* = $numexpr_3$.
    4. Set lvn = *A*.
    5. If lvn > *C*, processing of this forparameter is complete.
    6. Execute the commands in the scope of the FOR command once.
    7. If lvn > *C* ! *B*, processing of this forparameter is complete; an undefined value for lvn causes an error condition with ecode = "M15".
    8. Otherwise, set lvn = lvn + *B*.
    9. Go to 6.

c. If the forparameter is of the form $numexpr_1$: $numexpr_2$: $numexpr_3$ and $numexpr_2$ is negative.

    1. Set *A* = $numexpr_1$.
    2. Set *B* = $numexpr_2$.
    3. Set *C* = $numexpr_3$.
    4. Set lvn = *A*.
    5. If lvn < *C*, processing of this forparameter is complete.
    6. Execute the commands in the scope of the FOR command once.
    7. If lvn < *C* ! *B*, processing of this forparameter is complete; an undefined value for lvn causes an error condition with ecode = "M15".
    8. Otherwise, set lvn = lvn + *B*.
    9. Go to 6.

d. If the forparameter is of the form $numexpr_1$: $numexpr_2$.

    1. Set *A* = $numexpr_1$.
    2. Set *B* = $numexpr_2$.
    3. Set lvn = *A*.
    4. Execute the commands in the scope of the FOR command once.
    5. Set lvn = lvn + *B*; an undefined value for lvn causes an error condition with ecode = "M15".
    6. Go to 4.

If the FOR command has no argument:

a. Execute the commands in the scope of the FOR command once; since no lvn has been specified, it cannot be referenced.

b. Goto a.

Note that form d. and the argumentless form specify endless loops. Termination of these loops must occur by execution of a QUIT or GOTO command within the scope of the FOR command. These two termination methods are available within the scope of a FOR command independent of the form of forparameter currently in control of the execution of the scope of the FOR command; they are described below. Note

also that no forparameter to the right of one of form d. can be executed.

Note that if the scope of a FOR command (the *outer* FOR command) contains an *inner* FOR command, one execution of the scope of commands of the outer FOR command encompasses all executions of the scope of commands of the inner FOR command, corresponding to one complete pass through the inner FOR command's forparameter list.

Execution of a QUIT command within the scope of a FOR command has two effects.

a. When the QUIT command is executed, that particular execution of the scope of the FOR command is terminated at the QUIT command. I.e. commands to the right of the QUIT command are not executed.

b. After the QUIT command has been executed, any remaining values of the forparameter in control at the time of execution of the QUIT command, and the remainder of the forparameters in the same forparameter list, will not be calculated and the commands in the scope of the FOR command will not be executed under their control.

In other words, execution of a QUIT command causes the immediate termination of the innermost FOR command whose scope contains that QUIT command.

Execution of an argumented QUIT command within the scope of a FOR command causes an error condition with an ecode = "M16".

Execution of a GOTO command causes the immediate termination of all FOR commands in the line containing the GOTO command, and it transfers execution control to the point specified. Note that the execution of an argumentless QUIT command within the scope of a FOR command does not affect the local variable environment.

8.2.14 GOTO

G[OTO] postcond SP L gotoargument

gotoargument ::= | entryref postcond
@ expratom V L gotoargument |

GOTO is a generalized transfer of control. If provision for a return of control is desired, DO may be used.

Each gotoargument is examined, one at a time in left-to-right order, until the first one is found whose postcond is either absent, or whose postcond is present and its tvexpr is true. If no such gotoargument is found, control is not transferred and execution continues normally. If such a gotoargument is found, execution continues at the left of the line it specifies, provided that the following conditions hold for the line containing the GOTO command and the line specified by the gotoargument:

A. they have the same LEVEL, and

B. if that LEVEL is greater than one they

1. must have no lines of lower execution LEVEL between them, and
2. must be in the same routine.

If either A or B is not met, an error occurs with ecode = "M45".

8.2.15 HALT

H[ALT] postcond [SP]

If the value of $TLEVEL is greater then zero, a ROLLBACK is performed. In any case, all nrefs are removed from the LOCK-LIST associated with this process. Finally, execution of this process is terminated.

8.2.16 HANG

H[ANG] postcond SP L hangargument

hangargument ::= | numexpr
@ expratom V L hangargument |

Let *t* be the value of numexpr. If *t* '> 0, HANG has no effect. Otherwise, execution is suspended for *t* seconds.

For the possible effect of the use of non-integer values in the context of the HANG command, see clause 12 of Section 2, Portability Requirements.

8.2.17 IF

I[F] | [SP]
SP L ifargument |

ifargument ::= | tvexpr
@ expratom V L ifargument |

In its argumentless form, IF is the inverse of ELSE. That is, if the value of $TEST is 0, the remainder of the line to the right of the IF is not executed. If the value of $TEST is 1, execution continues normally at the next command.

If exactly one argument is present, the value of tvexpr is placed into $TEST; then the function described above is performed.

IF with *n* arguments is equivalent in execution to *n* IF commands, each with one argument, with the respective arguments in the same order. This may be thought of as an implied *and* of the conditions expressed by the arguments.

8.2.18 JOB

J[OB] postcond SP L jobargument

jobargument ::= | [jobenv] entryref [: jobparameters]
[jobenv] labelref actuallist [: jobparameters]
@ expratom V L jobargument |

jobenv ::= VB environment VB

jobparameters ::= | processparameters [timeout]
timeout |

processparameters ::= | expr
([[expr] :] ... expr) |

For each jobargument, the JOB command attempts to initiate another M[UMPS] process. If the actuallist is present in a jobargument, the formalline designated by labelref must contain a formallist in which the number of names is greater than or equal to the number of exprs in the actuallist.

The JOB command initiates this process at the line specified by the entryref or labelref. There is no linkage between the started process and the process that initiated it. It is erroneous for a jobargument to contain a call-by-reference actual (ecode = "M40"). If the actuallist is not present, the process will have no variables initially defined. (See 7.1.2.3 Process-Stack, and 8.1.7 Parameter passing).

The processparameters can be used in an implementation-specific fashion to indicate operational parameters.

If a timeout is present, the condition reported by $TEST is the success of initiating the process. If no timeout is present, the value of $TEST is not changed, and process execution is suspended until the process named in the jobargument is successfully initiated. The meaning of success in either context is defined by the implementation.

If jobenv is explicitly specified, the JOB command attempts to initiate this process in the environment specified by jobenv. Reference to a non-existent jobenv causes an error condition with an ecode = "M26". If jobenv is not explicitly specified, then the value of ^$JOB($JOB, "JOB") is used.

8.2.19 KILL

K[ILL] postcond killarglist

killarglist ::= | [SP]
SP L killargument |

killargument ::= | glvn
(L lname)
@ expratom V L killargument |

lname ::= | name
@ expratom V lname |

The three forms of KILL are given the following names.

a) glvn: Selective Kill.
b) (L lname): Exclusive Kill.
c) Empty argument list: Kill All.

KILL is defined using a subsidiary function K(*V, val, subs*) where *V* is a glvn, *val* is 1, and *subs* is 1.

a. Search for the name of *V* in the NAME-TABLE. If no such entry is found, the function is completed. Otherwise, extract the DATA-CELL pointer and proceed to step b.

b. In the DATA-CELL identified in step 'a', let N be the number of subscripts in *V*. If *V* is unsubscripted, let N be 0:

    1. If N is 0, then delete all tuples. The function is completed.

    2. Otherwise (if N > 0), delete all tuples of degree N or greater whose first N subscripts are the

same as those in *V*. The function is completed.

Note that as a result of procedure *K*(*V* , 1, 1), $DATA(*V*)=0, i.e., the value of *V* is undefined, and *V* has no descendants.

The actions of the three forms of KILL are then defined as:

a) Selective Kill — Apply procedure *K*(glvn , 1, 1).

b) Exclusive Kill — For all names, *V*, in the local variable NAME-TABLE except those in the argument list, apply procedure *K*(glvn , 1, 1). Note that the names in the argument list of an exclusive kill are restricted to unsubscripted locals.

c) Kill All — For all names, *V*, in the local variable NAME-TABLE, apply procedure *K*(glvn , 1, 1). Note that Kill All applies procedure *K* to the local variable NAME-TABLE only.

If a variable *N*, a descendant of *M*, is killed, the killing of *N* affects the value of $DATA(*M*) as follows: if *N* was not the only descendant of *M*, $DATA(*M*) is unchanged; otherwise, if *M* has a defined value $DATA(*M*) is changed from 11 to 1; if *M* does not have a defined value $DATA(*M*) is changed from 10 to 0.

8.2.20 KSUBSCRIPTS

KS[UBSCRIPTS] postcond killarglist

The three forms of KSUBSCRIPTS are given the following names.

a) glvn: Selective Kill.
b) (L lname): Exclusive Kill.
c) Empty argument list: Kill All.

KSUBSCRIPTS is defined using a subsidiary function K(*V, val, subs*) where *V* is a glvn, *val* is 0, and *subs* is 1.

a. Search for the name of *V* in the NAME-TABLE. If no such entry is found, the function is completed. Otherwise, extract the DATA-CELL pointer and proceed to step b.

b. In the DATA-CELL identified in step 'a', let N be the number of subscripts in *V*. If *V* is unsubscripted, let N be 0. Delete all tuples of degree N+1 or greater whose first N subscripts are the same as those in *V*. The function is completed.

Note that as a result of procedure *K*(*V* , 0, 1), $DATA(*V*)=1 if *V* had a value before procedure *K* was applied, or $DATA(*V*)=0 if *V* had no value before procedure *K* was applied, i.e., only the descendants of *V* are deleted.

The actions of the three forms of KILL are then defined as:

a) Selective Kill — Apply procedure *K*(glvn , 0, 1).

b) Exclusive Kill — For all names, *V*, in the local variable NAME-TABLE except those in the argument list, apply procedure *K*(glvn , 0, 1). Note that the names in the argument list of an exclusive kill are restricted to unsubscripted locals.

c) Kill All — For all names, *V*, in the local variable NAME-TABLE, apply procedure *K*(glvn , 0, 1). Note that Kill All applies procedure *K* to the local variable NAME-TABLE only.

If a variable *N*, a descendant of *M*, is killed, the killing of *N* affects the value of $DATA(*M*) as follows: if *N* was not the only descendant of *M*, $DATA(*M*) is unchanged; otherwise, if *M* has a defined value $DATA(*M*) is changed from 11 to 1; if *M* does not have a defined value $DATA(*M*) is changed from 10 to 0.

8.2.21 KVALUE

KV[ALUE] postcond killarglist

The three argument forms of KVALUE are given the following names.

a) glvn: Selective Kill.
b) (L lname): Exclusive Kill.
c) Empty argument list: Kill All.

KVALUE is defined using a subsidiary function K(*V, val, subs*) where *V* is a glvn, *val* is 1, and *subs* is 0.

a. Search for the name of *V* in the NAME-TABLE. If no such entry is found, the function is completed. Otherwise, extract the DATA-CELL pointer and proceed to step b.

b. If *V* is unsubscripted, delete the tuple of degree 0 (if found). The function is completed.

c. Otherwise, let N be the number of subscripts in *V*. Delete (if found) only the tuple of degree whose first N subscripts are the same as those in *V*. The function is completed.

Note that as a result of procedure *K*(*V* , 1, 0), $DATA(*V*)=0 if *V* had no descendants before procedure *K* was applied, or $DATA(*V*) = 10 if *V* had descendants before procedure *K* was applied, i.e., only the value of *V* is deleted.

The actions of the three forms KVALUE are then defined as:

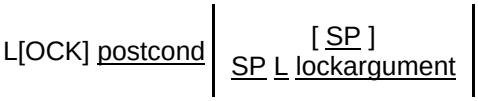
a) Selective Kill — Apply procedure *K*(glvn ,1 ,0).

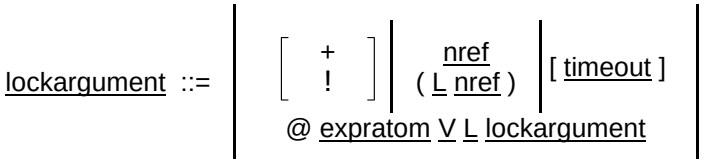
b) Exclusive Kill — For all names, *V*, in the local variable NAME-TABLE except those in the argument list, apply procedure *K*(glvn , 1, 0). Note that the names in the argument list of an exclusive kill are restricted to unsubscripted locals.

c) Kill All — For all names, *V*, in the local variable NAME-TABLE, apply procedure *K*(glvn ,1, 0). Note that Kill All applies procedure *K* to the local variable NAME-TABLE only.

If a variable *N*, a descendant of *M*, is killed, the killing of *N* affects the value of $DATA(*M*) as follows: if *N* was not the only descendant of *M*, $DATA(*M*) is unchanged; otherwise, if *M* has a defined value $DATA(*M*) is changed from 11 to 1; if *M* does not have a defined value $DATA(*M*) is changed from 10 to 0.

8.2.22 LOCK





```
nref  ::=  | rnref                          |
           | @ expratom V nref              |

rnref  ::= | [ ^ ] [ VB environment VB ] name [ ( L expr ) ]  |
           |          @ nrefind V ( L expr )                  |

nrefind  ::=  expratom V nref
```

LOCK provides a generalized interlock facility available to concurrently executing M[UMPS] processes to be used as appropriate to the applications being programmed. Execution of LOCK is not affected by, nor does it directly affect, the state or value of any global or local variable, or the value of the naked indicator. Its use is not required to access global variables, nor does its use inhibit other processes from accessing global variables. It is an interlocking mechanism whose use depends on programmers establishing and following conventions.

An nref is either unsubscripted or subscripted; if it is subscripted, any number of subscripts separated by commas is permitted.

When nrefind is present, it is always a component of a rnref. If the value of the rnref is a subscripted form of nref, then some of its subscripts may have originated in the nrefind. In this case, the subscripts contributed by the nrefind appear as the first subscripts in the value of the resulting rnref, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rnref.

Each lockargument specifies a subspace of the total M[UMPS] LOCK-UNIVERSE for the environment upon which the executing process seeks to make or release an exclusive claim; the details of this subspace specification are given below.

A special space for the lockspace is needed to create a synchronization mechanism for the executing process for each of the environments referenced by the executing process. A timeout refers to the time spent at the target environment, any time delays due to communication delays are not part of the timeout.

For the purposes of this discussion, the LOCK-UNIVERSE is defined as the union of all possible nrefs in one environment after resolution of all indirection. Further, there exists for each process a LOCK-LIST that contains zero or more nrefs. Execution of lockarguments has the effect of adding or removing nrefs from the process's LOCK-LIST. A given nref may appear more than once within the LOCK-LIST. The nrefs in the LOCK-LIST specify a subset of the LOCK-UNIVERSE. This subspace, called the process's LOCKSPACE, consists of the union of the subspaces specified by all nrefs in the LOCK-LIST, as follows:

a. If the nref is unsubscripted, then the subspace is the set of the following points: one point for the unsubscripted variable name nref and one point for each subscripted variable name $N(s_1, \ldots, s_i)$ where N has the same spelling as nref.

b. If the occurrence of nref is subscripted, let the nref be $N(s_1, s_2, \ldots, s_n)$. Then the subspace is the set of the following points: one point for $N(s_1, s_2, \ldots, s_n)$ and one point for each descendant (see 7.1.5.3 $DATA function for a definition of descendant) of nref. A subscript may not equal the empty string.

If the LOCK command is argumentless, LOCK removes all nrefs from the LOCK-LIST associated with the process executing the LOCK command.

Execution of lockargument occurs in the following order:

a. Any expression evaluation involved in processing the lockargument is performed.

b. If the form of lockargument does not include an initial + or ! sign, then prior to evaluating or

executing the rest of the lockargument, LOCK first removes all nrefs from the LOCK-LIST associated with the process executing the LOCK command. Then it appends each of the nrefs in the lockargument to the process's LOCK-LIST.

c. If the lockargument has a leading + sign, LOCK attempts to append each of the nrefs in the lockargument to the process's LOCK-LIST.

d. If the lockargument has a leading ! sign, then for each nref in the lockargument, if the nref exists in the LOCK-LIST for the process executing the LOCK command, one instance of nref is removed from the LOCK-LIST.

An error occurs, with ecode = "M41", if a process within a TRANSACTION attempts to remove from its LOCK-LIST any nref that was present when the TRANSACTION started. With respect to each other process, the effect of removing any nref from the LOCK-LIST is deferred until the global variable modifications made since that nref was added to the LOCK-LIST are available to that other process.

LOCK affects concurrent execution of processes having LOCK-SPACES that OVERLAP. Two LOCK-SPACEs OVERLAP when their intersection is not empty. LOCK imposes the following constraints on the concurrent execution of processes:

a. The LOCK-SPACEs of any two processes executing commands outside the scope of a TRANSACTION may not OVERLAP.

b. All global variable modifications produced by the execution of commands by processes having LOCK-SPACEs that OVERLAP must be equivalent to the modifications resulting from some execution schedule during which their LOCK-SPACEs do not OVERLAP.

See the TRANSACTION Processing subclause for the definition of TRANSACTION.

The constraints imposed by LOCK on the execution of processes having LOCK-SPACEs that OVERLAP may cause execution of one or more processes to be delayed. The maximum duration of such a delay may be specified with a timeout.

If present, timeout modifies the execution of LOCK, described above, as follows:

a. If execution of the process is delayed and cannot be resumed prior to the expiration of timeout, then the execution of the lockargument is unsuccessful. In this event the value of $TEST is set to zero and any nrefs added to the LOCK-LIST as a result of executing the lockargument are removed.

b. Otherwise, the execution of the lockargument is successful and $TEST is set to one.

If no timeout is present, then the value of $TEST is not affected by execution of the lockargument.

8.2.23 MERGE

M[ERGE] postcond SP L mergeargument

mergeargument ::= | $glvn_1$ = $glvn_2$ @ expratom V L mergeargument |

MERGE provides a facility to copy a $glvn_2$ into a $glvn_1$ and all descendants of $glvn_2$ into descendants of $glvn_1$ according to the scheme described below.

MERGE does not KILL any nodes in $glvn_1$, or any of its descendants.

Assume that $glvn_1$ is represented as $A(i_1 , i_2 , ... , i_x)$ (x '< 0) and that $glvn_2$ is represented as $B(j_1 , j_2 , ... , j_y)$ (y'<0). Then:

a. If $DATA(B($j_1$, j_2 , ... , j_y)) has a value of 1 or 11, then the value of glvn$_2$ is given to glvn$_1$.

b. The value for every occurrence of z, such that z > 0 and $DATA(B($j_1$, j_2 , ... , j_{y+z})) has a value of 1 or 11, the value of B(j_1 , j_2 , ... , j_{y+z}) is given to A(i_1 , i_2 , ... , i_x , j_{y+1} , j_{y+2} , ... , j_{y+z}).

The state of the naked indicator will be modified as if $DATA(glvn$_2$) # 10 = 1 and the command SET glvn$_1$ = glvn$_2$ would have been executed.

If glvn$_1$ is a descendant of glvn$_2$ or if glvn$_2$ is a descendant of glvn$_1$ an error condition occurs with ecode = "M19".

8.2.24 NEW

N[EW] postcond | [SP] SP L newargument |

newargument ::= | lname
newsvn
(L lname)
@ expratom V L newargument |

newsvn ::= | $ET[RAP]
$ES[TACK]
$R[EFERENCE]
$T[EST] |

NEW provides a means of performing variable scoping.

The four argument forms of NEW are given the following names:

a) lname: Selective NEW
b) (L lname): Exclusive NEW
c) Empty argument list: NEW All
d) newsvn NEW svn

The following discussion uses terms defined in the Variable Handling (see 7.1.2.2) and Process-Stack (see 7.1.2.3) models and, like those subclauses, does not imply a required implementation technique. Each argument of the NEW command creates a CONTEXT-STRUCTURE consisting of a NEW NAME-TABLE and an exclusive indicator, attaches it to a linked list of CONTEXT-STRUCTUREs associated with the current PROCESS-STACK frame, and modifies currently active NAME-TABLEs as follows:

a) NEW All — marks the CONTEXT-STRUCTURE as exclusive, copies the currently active NAME-TABLE to the NEW NAME-TABLE and makes all entries in the currently active local variable NAME-TABLE point to empty DATA-CELLs.

b) Exclusive NEW — marks the CONTEXT-STRUCTURE as exclusive, copies the currently active NAME-TABLE to the NEW NAME-TABLE and changes all entries in the currently active local variable NAME-TABLE, except for those corresponding to names specified by the command argument, to point to empty DATA-CELLs.

c) Selective NEW — copies the entry corresponding to the name specified by the command argument to the NEW NAME-TABLE and makes that entry in the currently active NAME-TABLE point to an empty DATA-CELL.

d) NEW svn — copies the entry corresponding to the name specified by the command argument to the NEW NAME-TABLE and updates that entry as follows:

1) if the argument specifies $ES[TACK], points to a DATA-CELL with a value of 0 (zero).

2) if the argument specifies $ET[RAP], points to a DATA-CELL with a value copied from the prior DATA-CELL (as pointed to by the just-copied NAME-TABLE entry).

3) if the argument specifies $R[EFERENCE], points to a DATA-CELL with a value copied from the priod DATA-CELL (as pointed to by the just-copied NAME-TABLE entry)

4) if the argument specifies $T[EST], points to a DATA-CELL with a value copied from the prior DATA-CELL (as pointed to by the just-copied NAME-TABLE entry).

8.2.25 OPEN

O[PEN] postcond SP L openargument

openargument ::= | devn [: openparameters] | @ expratom V L openargument |

openparameters ::= | deviceparameters [timeout [: mnemonicspec]] | [deviceparameters] :: mnemonicspec | timeout [: mnemonicspec] |

mnemonicspec ::= | mnemonicspace | (L mnemonicspace) |

mnemonicspace ::= expr V mnemonicspacename

mnemonicspacename ::= | ident [ident \| digit \| . \| !] ... | ^ routineref$_1$ [^ routineref$_2$] |

(Note: period)
(Note: hyphen)

There is a large overlap in specification between the commands OPEN, USE, and CLOSE. As a side-effect of the alphabetical ordering of the commands, many features are described in clause 8.2.6, the CLOSE command. As a matter of style in this document, these features are not repeated in this clause.

mnemonicspace specifies the set of controlmnemonics that may be used within format arguments to subsequent READ and WRITE commands. The mnemonicspace may be an empty string and may not provide any defined controlmnemonics. mnemonicspacenames that start with any character other than "Y" or "Z" are reserved for mnemonicspace definitions registered by the MDC; those that start with "Z" are implementor-specific.

The ^routineref alternative is a user-defined mnemonicspace and associates the routine named in routineref$_1$ with the location of code to be executed when a controlmnemonic is used.

The user-defined mnemonispace command routine is the routine defined in routineref$_2$, or if absent in

routineref$_1$. It associates this routine with the location of code to be executed when a command is used in conjunction with the mnemonicspace.

If an implementation does not provide for the use of a specific mnemonicspace then that implementation shall provide a mechanism by which to associate a routineref with this mnemonicspace. All subsequent references to this mnemonicspace are handled as if this were a user-defined mnemonicspace.

When a mnemonicspec contains a list of mnemonicspaces, the first one determines the active mnemonicspace. Subsequent USE commands may change the active mnemonicspace to any other mnemonicspace that is specified in this list.

If the device does not support a mnemonicspace in a mnemonicspec, an error condition occurs with ecode = "M35". If any mnemonicspaces in the mnemonicspec are incompatible, an error condition occurs with ecode = "M36".

In addition to controlmnemonics a mnemonicspace also defines the valid deviceattributes and devicekeywords which are associated with a device. deviceattributes and devicekeywords which start with the character “Z” are implementor-specific. Associated with each deviceattribute are one or more values which are held in the ssvn ^$DEVICE.

See 8.2.7 for the syntax and interpretation of devn and deviceparameters.

The OPEN command is used to obtain ownership of a device, and does not affect which device is the current device or the value of $IO. (see the discussion of USE in 8.2.35)

For each openargument, the OPEN command attempts to seize exclusive ownership of the specified device. OPEN performs this function effectively instantaneously as far as other processes are concerned; otherwise, it has no effect regarding the ownership of devices and the values of the device parameters. If a timeout is present, the condition reported by $TEST is the success of obtaining ownership. If no timeout is present, the value of $TEST is not changed and process execution is suspended until seizure of ownership has been successfully accomplished by the process that issued the OPEN command.

In the case that a process has successfully executed an OPEN command for a certain device and has established certain operational parameters for that device, and subsequently the same process makes an attempt to execute an OPEN command for the same device while specifying different operational parameters, those established operational parameters that are controlled by the implementation, and for which new values are supplied, will be discarded, and an attempt will be made to establish the newly specified parameters as the current ones for the device in question.

Ownership is relinquished by execution of the CLOSE command. When ownership is relinquished, all device parameters are retained. Upon establishing ownership of a device, any parameter for which no specification is present in the openparameters is given the value most recently used for that device; if none exists, an implementor-defined default value is used.

8.2.26 QUIT

Q[UIT] postcond	[SP] SP expr SP @ expratom V expr

QUIT terminates execution of an argumentless DO command, doargument, xargument, exfunc, exvar, or FOR command.

Encountering the end-of-routine mark eor is equivalent to executing an unconditional argumentless QUIT command.

The effect of executing a QUIT command in the scope of a FOR command is fully discussed in 8.2.13.

Note the eor never occurs in the scope of a FOR command.

If an executed QUIT command is not in the scope of a FOR command, then it is in the scope of some argumentless DO command, doargument, xargument, exfunc, or exvar if not explicitly then implicitly, because the initial activation of a process, including that due to execution of a jobargument, may be thought of as arising from execution of a DO command naming the first executed routine of that process.

The effect of executing a QUIT command in the scope of an argumentless DO command, doargument, xargument, exfunc, or exvar is to restore the previous variable environment (if necessary), restore the value of $TEST (if necessary), restore the previous execution level, and continue execution at the location of the invoking argumentless DO command, doargument, xargument, exfunc, or exvar.

If an expr is present in the QUIT command and the return is not to an exfunc or exvar, an error condition occurs with ecode = "M16". If the expr is not present and the return is to an exfunc or exvar, an error condition occurs with ecode = "M17".

The following discussion uses terms defined in the Variable Handling (see 7.1.2.2) and Process-Stack (see 7.1.2.3) models and, like those subclauses, does not imply a required implementation technique.

Execution of a QUIT command occurs as follows:

a. If an expr is present, evaluate it. If the resulting value is a value of data type OREF, do not coerce this value into a value of data type MVAL. This value becomes the value of the invoking exfunc or exvar.

b. Remove the frame on the top of the PROCESS-STACK. If no such frame exists, then execute an implicit HALT.

c. If the PROCESS-STACK frame's linked list of CONTEXT-STRUCTUREs contains NEW NAME-TABLEs, process them in last-in-first-out order from their creation. If the CONTEXT-STRUCTURE is exclusive, make all entries in the currently active local variable NAME-TABLE point to empty DATA-CELLs. In all cases, the NEW NAME-TABLEs are copied to the currently active NAME-TABLEs. Note that, in the model, QUIT never encounters any restart CONTEXT-STRUCTUREs in the linked list because they must have been removed by TCOMMITs or ROLLBACKs for the QUIT to reach this point in its execution.

d. If the frame contains formal list information:

    1. Extract the formallist and process each name in the list with the following steps:

        i. Search the NAME-TABLE for an entry containing the name. If no such entry is found, processing of this name is complete. Otherwise, proceed to step ii.

        ii. Delete the NAME-TABLE entry for this name.

    2. Finally, copy all NAME-TABLE entries from this frame into the NAME-TABLE.

    3. Processing of this frame is complete, continue at step b.

e. If the frame is a TSTART frame and $TLEVEL is greater than zero, QUIT generates an error with ecode = "M42". If the frame is a TSTART frame and $TLEVEL is zero, then the frame is discarded.

f. If the frame is from an exfunc or exvar or from an argumentless DO command, set the value of $TEST to the value saved in the frame. However, if this location is in a routine which has been modified or made inaccessible by the execution of a RSAVE command (subsequent to the placing of the frame on the PROCESS-STACK), unspecified behavior may result.

g. Restore the execution level and continue execution at the location specified in the frame.

8.2.27 READ

R[EAD] postcond SP L readargument

readargument ::= | strlit
format
glvn [readcount] [timeout]
* glvn [timeout]
@ expratom V L readargument |

readcount ::= # intexpr

The readarguments are executed, one at a time, in left-to-right order.

The forms strlit and format cause output operations to the current device; the forms glvn and *glvn cause input from the current device to the named variable (see 7.1.2.4 for a description of the value assignment operation). If no timeout is present, execution will be suspended until the input message is terminated, either explicitly or implicitly with a readcount. (See 8.2.35 for a definition of *current device*.)

If a timeout is present, it is interpreted as a *t* -second timeout, and execution will be suspended until the input message is terminated, but in any case no longer than *t* seconds. If *t* '> 0, *t* = 0 is used.

When a timeout is present, $TEST is affected as follows. If the input message has been terminated at or before the time at which execution resumes, $TEST is set to 1; otherwise, $TEST is set to 0.

When the form of the argument is *glvn [timeout], the input message is by definition one character long, and it is explicitly terminated by the entry of one character, which is not necessarily from any standardized character set. The value given to glvn is an integer; the mapping between the set of input characters and the set of integer values given to glvn may be defined by the implementor in a device-dependent manner. If timeout is present and the timeout expires, glvn is given the value !1.

When the form of the argument is glvn [timeout], the input message is a string of arbitrary length which is terminated by an implementor-defined procedure, which may be device-dependent. If timeout is present and the timeout expires, the value given to glvn is the string entered prior to expiration of the timeout; otherwise, the value given to glvn is the entire string.

When the form of the argument is glvn # intexpr [timeout], let *n* be the value of intexpr. If *n* '> 0 an error condition occurs with ecode = "M18". Otherwise, the input message is a string whose length is at most *n* characters, and which is terminated by an implementor-defined, possibly device-dependent procedure, which may be the receipt of the *n* th character. If timeout is present and the timeout expires prior to the termination of the input message by either mechanism just described, the value given to glvn is the string entered prior to the expiration of the timeout; otherwise, the value given to glvn is the string just described.

When it has been specified that the current device is able to send control-sequences according to some mnemonicspace, the READ command will be terminated as soon as such a control-sequence has been entered (be it by typing a function-key or by some other internal process within the device). The value of the specified glvn will be the same as if instead of the control-sequence the usual terminator-character would have been received before the control-sequence was sent.

When the form of the argument is strlit, it is equivalent to WRITE strlit. When the form of the argument is format, it is equivalent to WRITE format.

$X and $Y are affected by READ the same as if the command were WRITE with the same argument list (except for timeouts and readcounts) and with each expr value in each writeargument equal, in turn, to the final value of the respective glvn resulting from the READ.

Input operations, except when the form of the argument is *glvn [timeout], are affected by the Character Set Profile input-transform. Output operations are affected by the Character Set Profile output-transform. (see 7.1.3.1 ^$CHARACTER)

8.2.28 RLOAD

RL[OAD] postcond SP L routineargument

routineargument ::= | routineref : glvn [: routineparameters] @ expratom V L routineargument |

routineparameters ::= | routineparam ([[routineparam] :] ... routineparam) |

routineparam ::= | routinekeyword routineattribute = expr |

routinekeyword ::= name

routineattribute ::= name

Spellings of routinekeyword and routineattribute differing only in the use of lowercase and uppercase letters are equivalent.

All values of routinekeyword and routineattribute not starting with the character 'Z' are reserved for the MDC.

routinekeywords are processed in strict left-to-right order. When multiple equivalent routinekeywords are encountered, the last occurrence processed will define the action(s) to be taken.

Assume that glvn is represented as A(i_1 , i_2 , ... , i_x) (x '< 0). Then the lines of the routine denoted by routineref are stored in nodes A(i_1 , i_2 , ... , i_x , i_{x+1}). i_{x+1} has a value of *n* for the *n*th line of the routine for all lines of the routine, and no other nodes of A within the subscript range i_1 .. i_{x+1} will be affected.

The naked indicator is modified by the reference to glvn if it is a gvn, but not by the implicit reference to the immediate descendants of glvn.

If the routineref denotes a non-existent routine an error condition occurs with an ecode = "M88".

8.2.29 RSAVE

RS[AVE] postcond SP L routineargument

routinekeywords are processed in strict left-to-right order. When multiple equivalent routinekeywords are encountered, the last occurrence processed will define the action(s) to be taken.

Assume that glvn is represented as A(i_1 , i_2 , ... i_x) (x '< 0). Then the data values of all nodes A(i_1 , i_2 , ... , i_x , i_{x+1}) for which the value of $DATA is either 1 or 11 are stored as lines of the routine denoted by routineref. The lines are taken in the subscript ordering for i_{x+1} as specified in the definition of $ORDER (7.1.5.11).

If glvn is undefined or if no node A(i_1 , i_2 , ... , i_x , i_{x+1}) with a $DATA value of 1 or 11 exists, the routine denoted by routineref is deleted.

If any one of the values denoted by A(i_1 , i_2 , ... , i_x , i_{x+1}) does not conform to the definition of a line the effect of executing the RSAVE command is unspecified.

At no point during the execution of the RSAVE command will any process be able to see a partially-filed routine.

Execution of a RSAVE command where routineref names the currently-executing routine causes an error with ecode = "M25", and the routine is not modified.

The naked indicator is modified by the reference to glvn if it is a glvn, but not by the implicit reference to the immediate descendants of glvn.

8.2.30 SET

S[ET] postcond SP L setargument

setargument ::= | setdestination = expr
@ expratom V L setargument |

setdestination ::= | setleft
(L setleft) |

setleft ::= | leftrestricted
leftexpr
glvn
owproperty |

leftrestricted ::= | $D[EVICE]
$K[EY]
$R[EFERENCE]
$X
$Y |

leftexpr ::= | setpiece
setextract
setev
setqsub |

setpiece ::= $P[IECE] (glvn , $expr_1$ [, $intexpr_1$ [, $intexpr_2$]])

setextract ::= $E[XTRACT] (glvn [, $intexpr_1$ [, $intexpr_2$]])

setev ::= | $EC[ODE]
$ET[RAP] |

setqsub ::= $QS[UBSCRIPT] (glvn , intexpr)

Editor's note:
glvn doesn't seem right to me here: of course, since the value needs to be stored some place, the destination must be a glvn, but before the assignment occurs, the value of that glvn will need to be a namevalue.
Suggest to change glvn here to glvn V namevalue.

setdextract ::= $DE[XTRACT] (extracttemplate , L [recordfieldglvn])

setdpiece ::= $DP[IECE] (piecedelimiter , L [recordfieldglvn])

recordfieldglvn ::= glvn [: fieldindex]

Editor's note:
In recordfieldglvn, is that colon inside or outside the brackets?

See 7.1.2 for the definition of glvn. See 7.1.4.6 for the definition of intexpr.

SET is the general means both for explicitly assigning values to variables, and for substituting new values in pieces of a variable. Each setargument computes one value, defined by its expr. That value is then either assigned to each of one or more variables, or it is substituted for one or more pieces of a variable's current value. Each variable is named by one glvn.

Each setargument is executed one at a time in left-to-right order. The execution of a setargument occurs in the following order.

a. One of the following two operations is performed:

    1. If the portion of the setargument to the left of the = consists of one or more glvns, the glvns are scanned in left-to-right order and all subscripts are evaluated, in left-to-right order within each glvn.

    2. If the portion of the setargument to the left of the = consists of a setpiece or a setextract or a setqsub, the glvn that is the first argument of the setpiece or setextract or setqsub is scanned in left-to-right order and all subscripts are evaluated in left-to-right order within the glvn, and then the remaining arguments of the setpiece or setextract or setqsub are evaluated in left-to-right order.

b. The expr to the right of the = is evaluated. For each setleft, if it is a leftrestricted, the value to be assigned or replaced is truncated or converted to meet the inherent restrictions for that setleft before the assignment takes place. This means that in one SET command, the various setlefts may receive different values. If, however, a leftrestricted is either $X or $Y, the following additional considerations apply:

    1. The intexpr to the right of the = is evaluated.

    2. The value of the intexpr is given to the special intrinsic variable on the left of the = with the following restrictions and affects:

        a. The range of values of $X and $Y are defined in 7.1.4.10. Any attempt to set $X or $Y outside this range specified in 7.1.4.10 is erroneous (ecode = "M43") and the value of $X or $Y will remain unchanged.

        b. Setting $X or $Y changes the value of $X or $Y, respectively, but it does not cause any input or output operation. The purpose is to allow a program to correct the value of $X or $Y following input or output operations whose effect on the cursor position may not be reflected in $X and $Y.

c. One of the following specific operations is performed.

    1. If the left-hand side of the set is one or more glvns, the value of expr is given to each glvn, in left-to-right order. (See 7.1.2.2 for a description of the value assignment operation).

    2. For each setleft that is a setpiece, of the form $PIECE(glvn , *d* , *m* , *n*), the value of expr replaces the m^{th} through the n^{th} pieces of the current value of the glvn, where the value of *d* is the piece delimiter. Note that both *m* and *n* are optional. If neither is present, then *m* = *n* = 1; if

only *m* is present, then *n* = *m.* If glvn has no current value, the empty string is used as its current value. Note that the current value of glvn is obtained just prior to replacing it. That is, the other arguments of setpiece are evaluated in left-to-right order, and the expr to the right of the = is evaluated prior to obtaining the value of glvn.

Let *s* be the current value of glvn, *k* be the number of occurrences of *d* in *s*, that is, *k* = max(0, $LENGTH(*s* , *d*) !1), and *t* be the value of expr. The following cases are defined, using the concatenation operator _ of 7.2.1.1:

a. *m* > *n* or *n* < 1
The glvn is not changed and does not change the naked indicator.

b. *n* '< *m* !1 > *k*
The value in glvn is replaced by *s*_F(*m*!1!*k*)_*t*, where F(*x*) denotes a string of *x* occurrences of *d*, when *x* > 0; otherwise, F(*x*) = "". In either case, glvn affects the naked indicator.

c. *m* !1 '> *k* < *n*
The value in glvn is replaced by $PIECE(*s* , *d* , 1, *m* !1) _ F(min(*m* !1, 1)) _ *t*.

d. Otherwise,
The value in glvn is replaced by
$PIECE(*s* , *d* , 1, *m* !1) _ F(min(*m* !1, 1)) _ *t* _ *d* _ $PIECE(*s* , *d* , *n* +1, *k* +1).

3. For each setleft that is a setextract of the form $EXTRACT(glvn , *m* , *n*), the value of expr replaces the *m* th through the *n* th characters of the current value of the glvn. Note that both *m* and *n* are optional. If neither is present, then *m* = *n* = 1; if only *m* is present, then *n* = *m*. If glvn has no current value, the empty string is used as its current value. Note that the current value of glvn is obtained just prior to replacing it. That is, the other arguments of setextract are evaluated in left-to-right order, and the expr to the right of the = is evaluated prior to obtaining the value of glvn.

Let *s* be the current value of glvn, *k* be the number of characters in *s*, that is, *k* = $LENGTH(*s*), and *t* be the value of expr. The following cases are defined, using the concatenation operator _ of 7.2.1.1:

a. *m* > *n* or *n* < 1
The glvn is not changed and does not change the naked indicator.

b. *n* '< *m* !1 > *k*
The value in glvn is replaced by *s* _ $JUSTIFY("", *m* !1 - *k*) _ *t*.

c. *m* !1 '> *k* < *n*
The value in glvn is replaced by $EXTRACT(*s*, 1, *m* !1) _ *t*.

d. Otherwise,
The value in glvn is replaced by $EXTRACT(*s* , 1, *m* -1) _ *t* _ $EXTRACT(*s* , *n* +1, *k*).

In cases b), c) and d) the naked indicator is affected.

4. If the left-hand side of the SET is a setev, one of the following two operations is performed:

a. If the setev is $ECODE:

If the value of expr is the empty string:

1. The current value of $ECODE is replaced by the empty string.

2. All forms of the two-argument function $STACK($STACK + *n* , ...) return the empty string for all values of *n* > 0.

3. All forms of the function $STACK($STACK + *n*) return the empty string for all values of *n* > 0.

If the value of expr is not the empty string:

1. If the value of expr does not conform to the format required in section 7.1.4.10.2 for $ECODE, the SET of $ECODE to the value of the expr is not performed. Instead, an error condition occurs with ecode = "M101".

2. If the value of expr does conform to the format required in section 7.1.4.10.2 for $ECODE:

   a. The current value of $ECODE is replaced by the value of expr.

   b. The value of $STACK($STACK, "ECODE") is replaced by the value of expr.

   c. The value of $STACK($STACK, "PLACE") is replaced to reflect the SET command that is updating $ECODE.

   d. The value of $STACK($STACK, "MCODE") is replaced to reflect the SET command that is updating $ECODE.

   e. An error trap is invoked.

b. If the setev is $ETRAP, the current value of $ETRAP is replaced by the value of expr.

5. For each setleft that is a setgsub of the form $QSUBSCRIPT(*nv* , *m*), if the value of *nv* is not a valid namevalue, an error condition occurs with ecode = “M90". Otherwise, let *t* be the value of expr and *nv* in the form NAME(s_1 , s_2 , ... , s_n), considering *n* to be zero if there are no subscripts. The setleft is modified according to the value of intexpr *m* as follows:

   a. Values of *m* less than !1 are reserved for possible future use by the MDC.

   b. If *m* = !1, the environment is changed to *t*.

   c. If *m* = 0, the name is changed to *t*.

   d. If *m* > *n*, the intervening *n* + 1 through *m* - 1 subscripts are each set to the empty string and the m^{th} subscript is set to *t*.

   e. Otherwise, the m^{th} subscript is changed to *t*.

   If the resulting value of *nv* is not a valid namevalue, an error condition occurs with ecode = “M90".

   Note that the original and resulting namevalues are not “executed”, and will not modify the naked indicator beyond those modifications described at the end of this clause. Note also that the namevalues, while meeting the syntax of a namevalue, might specify a non-existent environment or contain a subscript value (such as the empty string or control characters) which does not meet the requirements of Section II Clause 2.3.3 (Values of subscripts).

6. If the left-hand side of the SET is an owproperty, the value of the expr is given to the owproperty.

7. For each setleft that is a setdextract, the expr is used as the starting value, which is partitioned

into consecutive $EXTRACT fields using extracttemplate (see page 49, $DEXTRACT). Each glvn is assigned its corresponding field extracted from expr. The values corresponding to omitted glvns are ignored. The fieldindex specifies which field is to be assigned to the glvn. If omitted, the next successive field index is assigned. Allthough all elements of the list of recordfieldglvns are optional, at least one recordfieldglvn (not necessarily the first) in the list must be non-empty.

8. For each setleft that is a setdpiece, the expr is used as the starting value, which is partitioned into consecutive $PIECE fields using piecedelimiter (see page 50, $DPIECE). Each glvn is assigned its corresponding field pieced from expr. The values corresponding to omitted glvns are ignored. The fieldindex specifies which field is to be assigned to the glvn. If omitted, the next successive field index is assigned. Allthough all elements of the list of recordfieldglvns are optional, at least one recordfieldglvn (not necessarily the first) in the list must be non-empty.

The value of the naked indicator may be modified as a side-effect of the execution of a SET command. Events that influence the value of the naked indicator are (in order of evaluation):

1. references to glvns in exprs in parameters or subscripts of setlefts;

2. references to glvns in the expr on the righthand side of the = sign;

3. references to glvns in the setdestination.

References that are defined as *scanned* in this clause do not affect the naked indicator, whereas references defined as *evaluated* do.

8.2.31 TCOMMIT

TC[OMMIT] postcond [SP]

If $TLEVEL is one, a TCOMMIT command performs a COMMIT of the TRANSACTION and sets $TRESTART to zero. (See the Transaction Processing subclause for the definition of COMMIT).

If $TLEVEL is greater than one, a TCOMMIT command subtracts one from $TLEVEL.

IF $TLEVEL is zero, a TCOMMIT command generates an error with ecode = "M44".

Using the (model) linked list of RESTART CONTEXT-STRUCTUREs for the TRANSACTION, a TCOMMIT command removes the last created RESTART CONTEXT-STRUCTURE from both the PROCESS-STACK linked list and the TRANSACTION linked list and discards the RESTART CONTEXT-STRUCTURE.

8.2.32 THEN

TH[EN] [SP]

This command creates a new CONTEXT-STRUCTURE consisting of a NEW NAME-TABLE and attaches it to a linked list of CONTEXT-STRUCTUREs associated with the current PROCESS-STACK frame, and modifies currently active NAME-TABLEs as per “NEW svn” for the svn $TEST. The value of $TEST is restored from this CONTEXT-STRUCTURE (and the CONTEXT-STRUCTURE is removed unless otherwise indicated) by any of the following actions (note the term “current line” refers to the line containing the THEN command):

- Execution encounters the eol at the end of the current line.
- A QUIT command is encountered at the current execution level (note: this includes RESTARTs
- A QUIT command returns execution to the current execution level (but does not remove the CONTEXT-STRUCTURE)
- An explicit GOTO command, located in the current line, is executed.

(Note: an Error Processing transfer of control (see 6.3) does not restore the value of $TEST.)

8.2.33 TRESTART

TRE[START] postcond [SP]

If $TLEVEL is greater than zero, a TRESTART command performs a RESTART.

If $TLEVEL is zero, a TRESTART command generates an error with ecode = "M44".

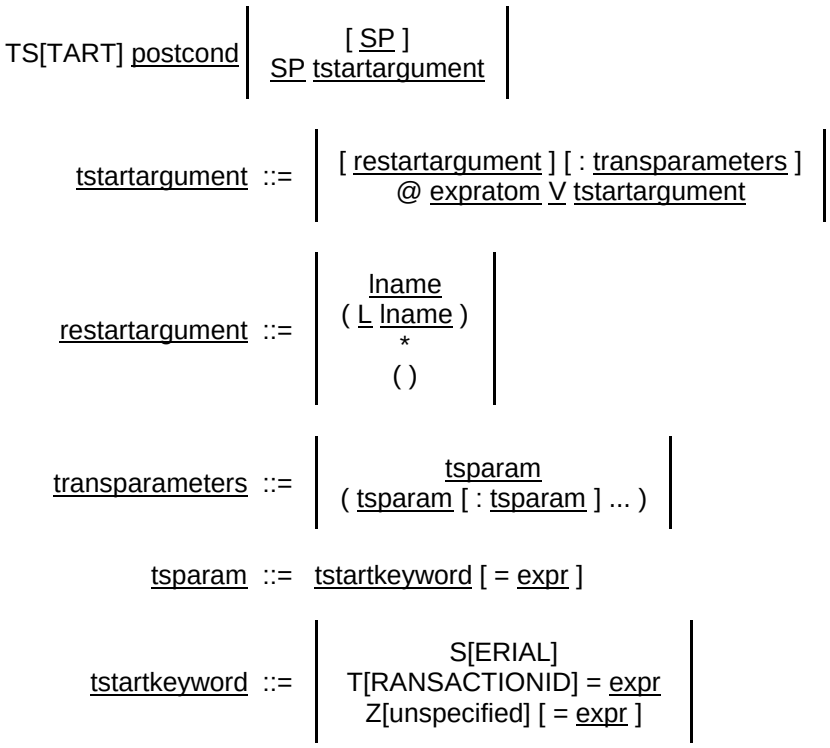
8.2.34 TROLLBACK

TRO[LLBACK] postcond [SP]

If $TLEVEL is greater than zero, a TROLLBACK command performs a ROLLBACK and sets $TLEVEL and $TRESTART to zero, and makes the naked indicator undefined. (See the Transaction Processing subclause for the definition of ROLLBACK).

If $TLEVEL is zero, a TROLLBACK command generates an error with ecode = "M44".

8.2.35 TSTART



tstartkeywords that differ only in the use of corresponding upper and lower-case letters are equivalent.

Unused keywords other than those starting with the letter "Z" are reserved for future extensions to the standard.

tstartkeywords are processed in strict left-to-right order. When multiple equivalent tstartkeywords are encountered, the last occurrence will define the action(s) to be taken.

After evaluation of postcond, if any, and tstartargument, if any, a TSTART command adds one to $TLEVEL. If, as a result, $TLEVEL is one, then a TSTART command initiates a TRANSACTION that is

restartable if a restartargument is present, or non-restartable if restartargument is absent; and serializable independently of LOCKs if transparameters are present and contain the keywords SERIAL or S, or dependent on LOCKs for serialization if those keywords are absent.

The tsparam, TRANSACTIONID, provides a means for identifying arbitrary classes of TRANSACTIONs.

The following discussion uses terms defined in the Variable Handling (see 7.1.2.2) and Process-Stack (see 7.1.2.3) models and, like those subclauses, does not imply a required implementation technique. The TSTART command creates a RESTART CONTEXT-STRUCTURE containing the execution location of the TSTART command, values for $TEST and the naked indicator, a copy of the process LOCK-LIST, a RESTART NAME-TABLE and an exclusive indicator. The TSTART command attaches the CONTEXT-STRUCTURE to a linked list of such RESTART CONTEXT-STRUCTUREs for the current TRANSACTION and also to a linked list of CONTEXT-STRUCTUREs associated with the current PROCESS-STACK frame. The TSTART command copies from the currently active NAME-TABLE to the RESTART NAME-TABLE all entries corresponding to the local variable names specified by the restartargument. The TSTART command also points the entries in the RESTART NAME-TABLE to copies of VALUE-TABLE tuples containing values that persist unchanged from the point that the TSTART command created the NAME-TABLE. When the restartargument is an asterisk (*), it specifies all current names and causes the CONTEXT-STRUCTURE to be marked as exclusive.

8.2.36 USE

U[SE] postcond SP L useargument

There is a large overlap in specification between the commands OPEN, USE, and CLOSE. As a side-effect of the alphabetical ordering of the commands, many features are described in clause 8.2.6, the CLOSE command. As a matter of style in this document, these features are not repeated in this clause.

Before a device can be employed in conjunction with an input or output data transfer it must be designated, through execution of a USE command, as the *current device*. Before a device can be named in an executed useargument, its ownership must have been established through execution of an OPEN command. See 8..2.7 for the syntax and interpretation of devn and deviceparameters.

The specified device remains current until such time as a new USE command is executed. As a side effect of employing expr to designate a current device, $IO is given the value of expr contained in devn and $IOREFERENCE is given the value of devn. See 7.1.4.10.6 and 7.1.4.10.7 for any differences between $IO and $IOREFERENCE.

Specification of device parameters, by means of the exprs in deviceparameters, is normally associated with the process of obtaining ownership; however, it is possible, by execution of a USE command, to change the parameters of a device previously obtained.

Distinct values for $X and $Y are retained for each device. The special variables $X and $Y reflect those values for the current device. When the identity of the current device is changed as a result of the execution of a USE command, the values of $X and $Y are saved, and the values associated with the new current device are then the values of $X and $Y.

8.2.37 VIEW

V[IEW] postcond arguments unspecified

VIEW makes available to the implementor a mechanism for examining machine-dependent information. It is to be understood that routines containing the VIEW command may not be portable.

8.2.38 WRITE

W[RITE] postcond SP L writeargument

writeargument ::= format | expr | * intexpr | @ expratom V L writeargument

The writeargument**s** are executed, one at a time, in left-to-right order. Each form of argument defines an output operation to the current device.

When the form of argument is format, processing occurs in left-to-right order.

format ::= positionformat [/ controlmnemonic [(L expr)]]

positionformat ::= [nlformat | ffformat] ... [tabformat] | tabformat

nlformat ::= !

ffformat ::= #

tabformat ::= ? intexpr

controlmnemonic ::= ? ident [ident | digit] ...

The following describes the effect of specific characters when used in a format:

! causes a *new line* operation on the current device. Its effect is the equivalent of writing CR LF on a pure ASCII device. In addition, $X is set to 0 and 1 is added to $Y.

causes a *top of form* operation on the current device. Its effect is the equivalent of writing CR FF on a pure ASCII device. In addition, $X and $Y are set to 0. When the current device is a display, the screen is blanked and the cursor is positioned at the upper left-hand corner.

? intexpr
produces an effect similar to *tab to column intexpr*. If $X is greater than or equal to intexpr, there is no effect. Otherwise, the effect is the same as writing (intexpr ! $X) spaces. (Note that the leftmost column of a line is column 0.)

/ controlmnemonic [(expr [, expr] ...)]
produces an effect which is defined by the mnemonicspace which has been assumed by default or has been selected in a previous mnemonicspace specification with a USE command. The relevant control-function is indicated by means of the controlmnemonic which must be defined in the above-mentioned mnemonicspace. Possible parameters are given through the optional exprs. Controlmnemonics which start with the character "?" are implementor-specific.

The implementor may restrict the use of controlmnemonics in a device-dependent way. A reference to an undefined mnemonicspace or an undefined controlmnemonic is reflected in special variable $DEVICE.

When the form of argument is expr, the value of expr is sent to the device. The effect of this string at the device is defined by appropriate device handling.

When the form of the argument is * intexpr, one character, not necessarily from the ASCII set and whose code is the number represented in decimal by the value of intexpr, is sent to the device. The effect of this character at the device may be defined by the implementor in a device-dependent manner.

As WRITE transmits characters one at a time, certain characters or character combinations represent device control functions, depending on the identity of the current device. To the extent that the supervisory function can detect these control characters or character sequences, they will alter $X and $Y as follows.

graphic	: add 1 to $X
backspace	: set $X = max($X ! 1, 0)
line feed	: add 1 to $Y
carriage return	: set $X = 0
form feed	: set $Y = 0, $X = 0

When a format specification is interpreted and the effect would cause the 'physical' external equivalent of $X and $Y to be modified, this effect will be reflected as far as possible in the values of the special variables $X and $Y.

Output operations, except when the form of the argument is * intexpr, are affected by the Character Set Profile output-transform.

8.2.39 XECUTE

X[ECUTE] postcond SP L xargument

xargument ::= | expr postcond
@ expratom V L xargument |

XECUTE provides a means of executing M[UMPS] code which arises from the process of expression evaluation.

Each xargument is evaluated one at a time in left-to-right order. If the postcond in the xargument is present and its tvexpr is false, the xargument is not executed. Otherwise, if the value of expr is *x*, execution of the xargument is executed in a manner equivalent to execution of DO *y*, where *y* is the spelling of an otherwise unused label attached to the following two-line subroutine considered to be a part of the currently executing routine:

```
y       ls x eol
        ls QUIT eol
```

8.2.40 Z

Z[unspecified] arguments unspecified

All commandwords in a given implementation which are not defined in the standard are to begin with the letter Z. This convention protects the standard for future enhancement.

8.3 Device Parameters

8.3.1 Output timeout

For any mnemonicspace the implementation may define a device parameter that causes an error condition when an output-producing argument of a READ or WRITE command fails to complete execution within a specified time. If it is defined, the device parameter shall conform to this clause and to the related sections of 7.1.3.2.

This device parameter shall have the following form:

OUTTIMEOUT = numexpr

numexpr shall be interpreted as the value of a timeout (see 8.1.5). Should any subsequent output-producing argument of a READ or WRITE command to the device fail to complete execution within that time, then

a. the OUTSTALLED member of ^$DEVICE, described in 7.1.3.2, shall assume the value 1, and

b. an error with ecode = “M100" shall occur.

Output timeout shall not apply to a device when

a. no OUTTIMEOUT deviceparam has executed for the device, or

b. the value of numexpr in the most recent OUTTIMEOUT is non-positive.

An execution of an OUTTIMEOUT deviceparam shall replace any previous OUTTIMEOUT deviceparam for the device.

The CLOSE command shall

a. Set the value of the OUTTIMEOUT deviceparam to 0.
b. Set the value of the OUTTIMEOUT member of ^$DEVICE to 0.
c. Set the value of the OUTSTALLED member of ^$DEVICE to 0.

Note: this is an exception to the general specification of device parameters in 8.2.2.

Note: output timeout applies to the execution of READ or WRITE arguments, not to the delivery of data to a device.

9 Character Set Profile charset

A charset is a definition of the valid characters and their characteristics available to a process. The required characteristics for a fully defined charset are:

a. The character codes and their meaning
b. The definition of which character codes are valid in names
c. The available patcodes and their definitions
d. The collation order of character strings.

Note: a charset definition is not necessarily tied to any (natural) language and could be an arbitrary set of characters or a repertoire from another set, such as ISO 10646.

charset ::= descriptor

descriptor ::= | % ident || [descsep] | ident digit | | ...

descsep ::= | ! | (note: hyphen)
_ | (note: underscore)
% |
* |
. |
/ |
+ |
: |
$ |
! |
@ |

The definition of the contents of standardized charsets is in Annex A. Unused charset names beginning with the initial letter Y are available for usage by M[UMPS] programmers; those beginning with the initial letter Z are reserved for vendor-defined charsets; all other charset names are reserved for future extensions to the standard.

American National Standard for Information Systems - Programming Languages - M[UMPS]
Section 2: M[UMPS] Portability Requirements)

Introduction

Section 2 highlights, for the benefit of implementors and application programmers, aspects of the language that must be accorded special attention if M[UMPS] program transferability (i.e., portability of source code between various M[UMPS] implementations) is to be achieved. It provides a specification of limits that must be observed by both implementors and programmers if portability is not to be ruled out. To this end, implementors must meet or exceed these limits, treating them as a minimum requirement. Any implementor who provides definitions in currently undefined areas must take into account that this action risks jeopardizing the upward compatibility of the implementation, upon subsequent revision of the M[UMPS] Language Specification. Application programmers striving to develop portable programs must take into account the danger of employing “unilateral extensions” to the language made available by the implementor.

The following definitions apply to the use of the terms *explicit limit* and *implicit limit* within this document. An explicit limit is one which applies directly to a referenced language construct. Implicit limits on language constructs are second-order effects resulting from explicit limits on other language constructs. For example, the explicit command line length restriction places an implicit limit on the length of any construct which must be expressed entirely within a single command line.

1 Character Set

The character set used for routines and data is restricted to the Character Set Profile M (as defined in Annex A).

2 Expression elements

2.1 Names

Portable name length is limited to thirty-one (31) characters. All characters in a name are significant in determining uniqueness. Therefore the length restriction places an implicit limit on the number of unique names on an implementation. If a name's length exceeds an implementor's limit an error condition occurs with ecode = "M56".

2.2 External routines and names

The externalroutinename namespace is unspecified, as this is a function of the binding, although at the present time, a maximum of twenty-four (24) characters allowed is placed upon externalroutinenames to be treated uniquely, although this should be viewed as a minimum number that needs to be handled rather than as the maximum number that can be used. Any number of characters, from one to the maximum number shall be valid as externalroutinenames. Any additional external mapping between these names and any actually used by an external package is an implementation issue.

2.3 Local variables

2.3.1 Number of local variables

The number of local variable names in existence at any time is not explicitly limited. However, there are implicit limitations due to the storage space restrictions (Clause 8).

2.3.2 Number of subscripts

There is no explicit limit on the number of distinct local variable nodes which may be defined, but there is an implicit limit based on the number of subscripts that may be defined for any local variable reference. The number of subscripts in a local variable is limited in that, in a local array reference, the total length of the array reference must not exceed 510 characters. The length of an array reference, assuming it is in the form name (i_1 , i_2 , ... , i_n), is calculated as follows. If:

N = $LENGTH(name),

I = $LENGTH($i_1$) + $LENGTH($i_2$) + ... + $LENGTH($i_n$), where each subscript (i_1 through i_n) is either a numlit or a sublit, and

L = n,

then:

the total length of an array reference = N + I + (2 * L) + 15.

2.3.3 Values of subscripts

Local variable subscript values are non-empty strings which shall only contain characters from the M[UMPS] printable character subset. The length of individual subscripts is limited to 255 characters; in addition, a complete variable name reference is limited according to the restrictions specified in 2.3.2.

When the subscript value satisfies the definition of a numeric data value (See 7.1.4.3 of Section 1), it is further subject to the restrictions of number range given in 2.6. The use of subscript values which do not meet these criteria is undefined, except for the use of the empty string as the last subscript of a starting reference in the context of data transversal functions such as $ORDER and $QUERY.

2.4 Global variables

2.4.1 Number of global variables

There is no explicit limit on the number of distinct global variable names in existence at any time.

2.4.2 Number of subscripts

The number of subscripts in a global variable is limited in that, in a global array reference, the total length of the array reference must not exceed 510 characters. The length of an array reference, assuming it is in the form ^ VB environment VB name (i_1 , i_2 , ... , i_n), is calculated as follows. If:

E = $LENGTH(environment),

N = $LENGTH(name),

I = $LENGTH($i_1$) + $LENGTH($i_2$) + ... + $LENGTH($i_n$), where each subscript (i_1 through i_n) is either a numlit or a sublit, and

L = n,

then:

the total length of an array reference = E + 3 + N + I + (2 * L) + 15.

2.4.3 Values of subscripts

The restrictions imposed on the values of global variable subscripts are identical to those imposed on local variable subscripts (see 2.3.3).

2.4.4 Number of nodes

There is no explicit limit on the number of distinct global variable nodes which may be defined.

2.5 Data types

The M[UMPS] Language Specification defines two data types, namely, MVALs (variable length character strings) and OREFs (object references). Contexts which demand a numeric, integer, or truth value interpretation are satisfied by unambiguous rules for mapping an MVAL into a number, integer, or truth value.

The implementor is not limited to any particular internal representation. Any internal representation(s) may be employed as long as all necessary mode conversions are performed automatically and all external behavior agrees with the M[UMPS] Language Specification. For example, integers might be stored as binary integers and converted to decimal character strings whenever an operation requires a string value.

2.6 Number range

All values used in arithmetic operations or in any context requiring a numeric interpretation are within the inclusive intervals [-10^{25}, -10^{-25}] or [10^{-25}, 10^{25}], or are zero.

Implementations shall represent numeric quantities with at least 15 significant digits. The error introduced by any single instance of the arithmetic operations of addition, subtraction, multiplication, division, integer

division, or modulo shall not exceed one part in 10^{15}. The error introduced by exponentiation shall not exceed one part in 10^{7}.

If the result of any arithmetic operation is too large (either positive or negative), or if it is too large for the implementation to represent with the accuracy specified in the previous paragraph, an error condition occurs with ecode = "M92".

Programmers should exercise caution in the use of non-integer arithmetic. In general, arithmetic operations on non-integer operands or arithmetic operations which produce non-integer results cannot be expected to be exact. In particular, non-integer arithmetic can yield unexpected results when used in loop control or arithmetic tests.

2.7 Integers

The magnitude of the value resulting from an integer interpretation is limited by the accuracy of numeric values (see 2.6). The values produced by integer valued operators and functions also fall within this range (see 7.1.4.6 of Section 1 for a precise definition of integer interpretation).

2.8 Character strings

Character string length is limited to 32,767 characters for local variables, 510 characters for global variables, and 32,767 characters for structured system variables. The characters permitted within character strings must include those defined in the ASCII Standard (ANSI X3.4-1986). If a string's length exceeds an implementor's limit, an error condition occurs with ecode = "M75".

2.9 Special variables

If appending the information about a new error condition (See 6.3.2 of Section 1) to $ECODE or $STACK($STACK,"ECODE") would exceed an implementation's maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,"ECODE") to discard.

The value of $SYSTEM as provided by an implementor must conform to the requirements for a local variable subscript (see 2.3.3).

The special variables $X and $Y are non-negative integers (see 2.7). The effect of incrementing $X and $Y past the maximum allowable value is undefined. (For a description of the cases in which the values of $X and $Y may be altered see 8.2.35 of Section 1; for a description of the type of values $X and $Y may have see 7.1.4.10 of Section 1).

3 Expressions

3.1 Nesting of expressions

The number of levels of nesting in expressions is not explicitly limited. The maximum string length does impose an implicit limit on this number (see 2.8).

3.2 Results

Any final result that does not satisfy the constraints on character strings (see 2.8) is erroneous. Any intermediate result that does not satisfy the constraints on local variable character strings (see 2.8) is erroneous. Furthermore, integer results are erroneous if they do not satisfy the constraints on integers (see 2.7).

3.3 External References

External references are not portable.

4 Routines and command lines

4.1 Command lines

A command line (line) must satisfy the constraints on global variable character strings (see 2.8). The length of a command line is the number of characters in the line up to but not including the eol.

The characters within a command line are restricted to the 95 ASCII printable characters. The character set restriction places a corresponding implicit restriction upon the value of the argument of the indirection delimiter (Clause 7).

4.2 Number of command lines

There is no explicit limit on the number of command lines in a routine, subject to storage space restrictions (Clause 8).

4.3 Number of commands

The number of commands per line is limited only by the restriction on the maximum command line length (see 4.1).

4.4 Labels

A label of the form name is subject to the constraint on names (see 2.1), with the exception that the first 31 characters are uniquely distinguished. Labels of the form intlit are subject to the same length constraints.

4.5 Number of labels

There is no explicit limit on the number of labels in a routine. However, the following restrictions apply:

a) A command line may have only one label.

b) No two lines may be labeled with equivalent (not uniquely distinguishable) labels.

4.6 Number of routines

There is no explicit limit on the number of routines. The number of routines is implicitly limited by the name length restriction (see 2.1).

5 External routine calls

When the external routine called is not within the current default M[UMPS] environment, all variables should be assumed to be scalars (i.e., *a* refers to the value associated with *a*, but does not refer to any descendants *a* might have such as *a(1)*, et cetera.). No prohibition against non-scalar extensions should be inferred, only that they may not be portable. It should be noted that no all-encompassing implied guarantee of the number of routines supported by an external package exists.

6 Character Set Profiles

Character Set Profiles are registered through the MUMPS Development Committee (ANSI X11). New Character Set Profile Definitions are approved through the standard procedures of the MUMPS Development Committee.

Routines and data created using a registered Character Set Profile are portable to all implementations which support that Character Set Profile.

The list of MDC registered Character Set Profiles is included in Annex A.

Note that subscript-string length (see 2.3.2, 2.3.3, 2.4.2, 2.4.3) is either the length of the value of the subscript, or the length of the computed Character Set Profile collation value, whichever is larger.

Collation values are not portable between implementations unless the value is explicitly stated in the definition of the Character Set Profile.

7 Indirection

The values of the argument of indirection and the argument of the XECUTE command are subject to the constraints on character string length (see 2.8). They are additionally restricted to the character set limitations of command lines (see 4.1).

8 Storage space restrictions

The size of a single routine must not exceed 20,000 characters. The size of a routine is the sum of the sizes of all the lines in the routine. The size of each line is its length (as defined in 4.1) plus two.

Note: In contrast to previous versions of the standard, there no longer is a specification of local variable storage. Like global variable storage, local variable storage can be arbitrarily large. The implementation's conformance statement must specify the minimum guaranteed amount to be available.

9 Process-Stack

Systems will provide a minimum of 127 levels in the PROCESS-STACK. The actual use of all these levels may be limited by storage restrictions (Clause 8).

Nesting within an expression is not counted in this limit. Expression nesting is not explicitly limited; however, it is implicitly limited by the storage restriction (Clause 8).

10 Formats

Device control may be effected through the READ and WRITE commands using the /controlmnemonic syntax in a specification of a format. In general, portability of routines containing such syntax is only possible in cases which meet several criteria, most obviously

a. the devices to be used at the receiving facility must have all the capabilities required by the /controlmnemonic occurrences in the routines;

b. the implementors of the systems at both the originating and the receiving facilities have implemented each combination of mnemonicspace and controlmnemonic in compatible ways.

As a result of these limitations, 'blind interchange' will only be dependent upon the devices at the receiving site.

However, the following advice to both implementors and programmers will increase the number of cases in which 'informed interchange' will be possible.

However user-defined mnemonicspaces, together with their associated controlmnemonics, are inherently portable provided that the M[UMPS] routines are also portable.

10.1 mnemonicspace

For portability, the mnemonicspace to be used must be a generally accepted standard, e.g. ANSI X3.64 or GKS, or after such a standard would have been accepted, any other ANSI or ISO standard.

10.2 controlmnemonic

For portability, the controlmnemonic must be one of the controlmnemonics assigned to a control-function specified in the chosen mnemonicspace and interpretation of the format specification must lead to the effect described in the mnemonicspace. There should be no other (side-)effects on the device.

With regard to the status of the process, the value of some special variables may change, e.g. with some control-functions $X and $Y would have to receive proper values. Apart from these documented effects, no other effects may be caused by any implementation.

An implementation needs not to allow for all controlmnemonics in all mnemonicspaces.

10.3 Parameters

A format containing /controlmnemonic may contain one or more parameters, specified as L expr, in which case each expr specifies a parameter of the control-function. The exprs must appear in the same order and number as the parameters in the corresponding mnemonicspace. The value of each expr should meet the limitations of 2.6 through 2.8.

11 Transaction processing

11.1 Number of modifications in a TRANSACTION

The sum of the lengths of the namevalues and values of global variable tuples modified within a TRANSACTION must not exceed 57,343 characters.

11.2 Number of nested TSTARTs within a TRANSACTION

A single TRANSACTION must not contain more than 126 TSTARTs after the TSTART that initiates the TRANSACTION.

12 Event processing

12.1 Number of timers

The number of concurrently running timers must not exceed one (1) per process or sixteen (16) per system, whichever is smaller.

12.2 Depth of event queues

The per-process event queues (one each for synchronous and asynchronous events) must not contain more than one event.

12.3 Resolution of timers

Timers must not use a resolution finer than one second.

12.4 Event classes

Use of the following event classes may not be portable: COMM, INTERRUPT, POWER, and Z[unspecified]. Use of HALT event classes where evid does not equal 1 may not be portable.

13 Other portability requirements

Programmers should exercise caution in the use of non-integer values for the HANG command and in TIMER events and timeouts. In general, the period of actual time which elapses upon the execution of a HANG command, or which elapses before a TIMER event, cannot be expected to be exact. In particular, relying upon non-integer values in these situations can lead to unexpected results.

Implementations may restrict access to ssvns that contain default environments of processes other than the one referring to the ssvn. Therefore, portable programs shall not rely on the ssvns defined in 7.1.3.10 when processid is not their own $JOB.

American National Standard for Information Systems - Programming Languages - M[UMPS]
Section 3: X3.64 Binding

Introduction

ANSI X3.64 is a functional standard for additional control functions for data interchange with two-dimensional character-imaging input and/or output devices. It is an ANSI standard, but also an ISO standard with roughly similar characteristics exists (ISO 2022). As such, it has been implemented in many devices worldwide. It is expected that M[UMPS] can be easily adapted to these implementations.

The standard defined as ANSI X3.64 defines a format for device-control. No physical device is required to be able to perform all possible control-functions. In reality, as some functions rely on certain physical properties of specific devices, no device will be able to perform all functions. The standard, however, does not specify which functions a device should be able to do, but if it is able to perform a function, how the control-information for this function is to be specified.

This binding is to the functional definitions included in X3.64. The actual dialogue between the M[UMPS] implementation and the device is left to the implementor.

1 The binding

ANSI X3.64 is accessed from the M[UMPS] language by making use of mnemonicspaces. A controlmnemonic from X3.64 may be accessed as follows:

/controlmnemonic [(expr [, expr] ...)]

where the relevant controlmnemonic equals the name of the generic function and exprs the possible applicable parameters. The use of a controlmnemonic produces the effect defined in ANSI X3.64 for the control-function with the same name as the controlmnemonic specified.

Some controlmnemonics return a value, or a collection of values. It is perfectly legal to issue these controlmnemonics with either a READ or WRITE command. If a READ command is used, the argument list in the statement(s) must be ordered to correctly accept the returned values. If a WRITE command is used the values returned may be read by a single, or series of, READ commands. These READ commands must be correctly ordered to match the returned values, however there may be intermediate calculations utilizing some of the returned values before reading the remaining values in the list. Reading the return list of values may be terminated without error by issuing another controlmnemonic. In this case, all returned values not assigned to a variable will be lost to the application program.

All controlmnemonics have the same name in M[UMPS] as in X3.64.

Unless explicitly mentioned, the use of X3.64 controlmnemonics has no side-effects on special variables such as $X, $Y, $KEY and $DEVICE.

1.1 Control-functions with an effect on $X or $Y or both

Below follows a list of control-functions (X3.64) or controlmnemonics (M[UMPS]) that have an effect on the special variables $X or $Y or both. Since some definitions in X3.64 are fairly open-ended, the exact effect may be implementation dependent in some cases. In section 3.4 these open-ended definitions are listed resolution of possible ambiguities are stated.

The relevant controlmnemonics are:

/CBT(n)	$X
/CHA(x)	$X
/CHT(n)	$X
/CNL(n)	$X, $Y
/CPL(n)	$X, $Y
/CUB(n)	$X
/CUD(n)	$Y
/CUF(n)	$X
/CUP(y,x)	$X, $Y
/CUU(n)	$Y
/CVT(n)	$Y
/HPA(x)	$X
/HPR(n)	$X
/HTJ	$X
/HVP(y,x)	$X, $Y
/IND	$Y
/NEL	$X, $Y
/PLD	$Y
/PLU	$Y
/REP(n)	$X, $Y
/RI	$Y
/RIS	$X=0, $Y=0
/VPA(y)	$Y

/VPR(n) $Y

The control-function REP repeats the previous character or function as many times as indicated by its argument. Hence, the side-effects of this function do not depend on this function itself, but rather on the character or function that is being repeated.

1.2 Control-functions with an effect on $KEY

Currently only one controlmnemonic may have a side-effect on special variable $KEY: /DSR (device status report). The side-effect depends on the value of the parameter of this function: parameter-value 0 or 5 will cause a status report to be returned, parameter-value 6 will cause the active cursor-position to be returned. The format of the value returned is:

$CHAR(27,91)_REPORT_$CHAR(110)

or

$CHAR(27,91)_Y_$CHAR(59)_X_$CHAR(82)

where REPORT is a code for the status reported, Y is the value of the current Y-coordinate and X is the value of the current X-coordinate.

The values described will be reported in special variable $KEY as a side-effect of the first READ command that is executed after the control-function has been issued.

1.3 Control-functions with an effect on $DEVICE

All controlmnemonics will have a side-effect on special variable $DEVICE. The most common situation will be that $DEVICE will receive the value:

"0,,X3-64"

in order to reflect the correct processing of a controlmnemonic.

In certain situations a status has to be indicated. Status codes for $DEVICE relating to X3.64 are as follows:

code	American English Description
1	mnemonicspace not found
2	invalid mnemonic
3	parameter out of range
4	hardware error
5	mnemonic not available for this device
6	parameter not available for this device
7	attempt to move outside boundary - not moved
8	attempt to move outside boundary - moved to boundary
9	auxiliary device not ready

1.4 Open-ended definitions

Under some conditions, the behavior specified by X3.64 is either ambiguous or optional. The following clarifies the behavior to ensure consistency:

CBT Move the cursor to the last horizontal tabulator-stop in the previous line. If no such tabulator-stop exists, don't move the cursor.

CHA when a location outside the available horizontal range is specified:
Move the cursor in the direction suggested by the parameter-value to either the rightmost (parameter value greater than current position) or leftmost (parameter value less than current position) position.

CHT when no further forward horizontal tabulator-stops have been defined in the current line:
Move the cursor to the first horizontal tabulator-stop in the next line. If no such tabulator-stop exists, don't move the cursor.

CNL when the cursor is moved forward beyond the last line on the device:
Do not move the cursor. If the output device is a CRT-screen, scroll up one line.

CPL when the cursor is moved backward beyond the first line on the device:
Do not move the cursor. If the output device is a CRT-screen, scroll down one line.

CUB when the cursor is moved backward beyond the first position on a line:
Do not move the cursor.

CUD when the cursor is moved downward beyond the last line on a device:
Do not move the cursor.

CUF when the cursor is moved forward beyond the last position on a line:
Do not move the cursor.

CUP when a location outside the available horizontal or vertical ranges is specified:
Do not move the cursor.

CUU when the cursor is moved upward beyond the last line on a device:
Do not move the cursor.

CVT when no further forward vertical tabulator-stops have been defined on the device:
Move the cursor to the first vertical tabulator-stop in the next page. If no such tabulator-stop exists, don't move the cursor.

HPA when a location outside the available horizontal range is specified:
Move the cursor in the direction suggested by the parameter-value to either the rightmost (parameter value greater than current position) or leftmost (parameter value less than current position) position.

HPR when a location outside the available horizontal range is specified:
Move the cursor in the direction suggested by the parameter-value to either the rightmost (parameter value positive) or leftmost (parameter value negative) position.

HTJ when no further forward horizontal tabulator-stops have been defined in the current line:
Move the cursor to the first horizontal tabulator-stop in the next line. If no such tabulator-stop exists, don't move the cursor.

HVP when a location outside the available horizontal or vertical ranges is specified:
Do not move the cursor.

IND when the cursor is moved downward beyond the last line on a device:
Move the cursor to the corresponding horizontal position in the first line on the next page.

NEL when the cursor is moved downward beyond the last line on a device:
Move the cursor to the first position on the first line on the next page.

PLD this function may or may not be similar to CUD or IND. The effect of two successive PLD operations may or may not be equal to the effect of one single CUD or IND operation:
This function will be identical to CUD.
The effect of PLD and PLU will be complementary, i.e. .PLD immediately followed by PLU will effectively not move the cursor.

PLU this function may or may not be similar to CUU or RI. The effect of two successive PLU operations may or may not be equal to the effect of one single CUU or RI operation:
This function will be identical to CUU.
The effect of PLD and PLU will be complementary, i.e. .PLU immediately followed by PLD will effectively not move the cursor.

RI when the cursor is moved upward beyond the first line on a device:
Move the cursor to the corresponding horizontal position in the last line on the previous page.

VPA when a location outside the vertical range is specified:
Move the cursor in the direction suggested by the parameter-value to either the bottommost (parameter value greater than current position) or topmost (parameter value less than current position) position.

VPR when a location outside the vertical range is specified:
Move the cursor in the direction suggested by the parameter-value to either the bottommost (parameter value positive) or topmost (parameter value negative) position.

The following functions shall not cause the cursor to move: ICH, JFY, MC, NP, DL and PP.

The following functions shall move the cursor so that it will point to the same character in the new projection of the information: SD, SL, SR and SU. Boundary conditions will be similar to CUD, CUB, CUF and CUU respectively.

2 Portability issues

2.1 Implementation

Any implementation of this binding shall accept all controlmnemonics specified. However, in most cases all controlmnemonics will not be supported for all devices. The appropriate error code will be returned in $DEVICE to indicate if a particular controlmnemonic is supported for the current device.

2.2 Application

Several controlmnemonics specified in X3.64 are ambiguous and usage of these will likely have different meaning between different devices and implementations. Usage of these will not be portable.

Control-mnemonic	Control Function
APC	Application Program Command
DA	Device Attributes
DCS	Device Control String
FNT	Font Selection
INT	Interrupt
OSC	Operating System Command
PLD	Partial Line Down (CUD recommended; see 1.4)
PLU	Partial Line Up (CUU recommended; see 1.4)
PM	Privacy Message
PU1	Private Use One
PU2	Private Use Two
SGR	Select Graphic Rendition for the following:
	10 primary font
	11 first alternative font
	12 second alternative font
	13 third alternative font
	14 fourth alternative font
	15 fifth alternative font
	16 sixth alternative font
	17 seventh alternative font
	18 eighth alternative font
	19 ninth alternative font
SS2	Single Shift Two
SS3	Single Shift Three

3 Conformance

Each implementation must supply a list of the controlmnemonic and arguments that are supported for each device.

Annex A: Character Set Profiles (normative)

The definition of a Character Set Profile requires the definition of four elements)) the names of the characters in the character set and the internal codes which are used to represent them, the definitions of which characters match which pattern codes, the collation scheme used, and the definition of which characters may be used in names.

Note that the patcodes A, C, E, L, N, P, and U are applicable for all character set profiles; in addition patcode E matches any character, not just those listed in any specific charset.

Two collation schemes are provided which only require a properly defined table of characters for the Character Set associated with the specific Character Set Profile.

STRING COLLATION

Determining the Collation Ordering for a Character Set Profile requires the collation value(s) for each character within the character set be accessible a group of values presented as an *n*-tuple. Each column of the definition table provides one value of the tuple in the specified order. When no value is present in any column, the corresponding character ID value is used in its place. Note that certain characters may be represented with more than one value entry line in the table; in these cases the entries are taken one at a time and treated as if they represented separate characters in the original string (e.g., the character *Æ* in ISO-Latin ! 1 (Character ID number 198) would be treated as a form of the string "AE").

Let *s* be any non-empty string. Define the numeric function $CV_n(s)$ to return the *n*th-order collation value for string *s*: unless otherwise specified this value is determined by evaluating the value in the *n*th column of each collation tuple for each character in the string examined in left-to-right order and combining them together. Note: selected collation-tuple columns may optionally be designated for right-to-left evaluation.

The Collation Ordering function CO determines relative ordering for a character set. The exact value of this function is not specified here, however, the values formed by any implementation must satisfy the following rules when comparing two non-equal strings:

> Let *t* also be any non-empty string, not equal to *s*. The *STRING* Collation Ordering function CO is defined as:

a. CO("" , *s*) = *s*

b. CO(*s* , *t*) = *t*
if, and only if, there is a *j* such that $CV_j(t) > CV_j(s)$
and for all i, i=1 ... j ! 1, $CV_i(t) = CV_i(s)$;
otherwise CO(*s*,*t*) = *s*.

M[UMPS] COLLATION

The *M[UMPS]* Collation Ordering function CO uses the definition of $CV_n(s)$ specified in *STRING* Collation and is otherwise different only with respect to numbers:

> Let *s* be any non-empty string, let *m* and *n* be strings satisfying the definition of numeric data values (see I.7.1.4.3), and *u* and *v* be non-empty strings which do not satisfy that definition.

a. CO("" , *s*) = *s*

b. CO(*m* , *n*) = *n*
if *n* > *m*; otherwise, CO(*m* , *n*) = *m*

c. CO(*m* , *u*) = *u*

d. CO(*u* , *v*) = *v*
if, and only if, there is a *j* such that $CV_j(v) > CV_j(u)$
and for all i, i=1 ... j ! 1, $CV_i(v) = CV_i(u)$;
otherwise, CO(*u*,*v*)=*u*.

1 charset M

The charset M is defined using the table A.1. The values in the columns headed Character ID and Character Symbol are taken from ASCII (X3.4-1990). The column headed patcode defines which characters match the patcodes A, C, E, L, N, P, and U. The characters in the table with a patcode of A are defined as idents. The collation rule used is *M[UMPS]* collation, using the collation order values provided in the table.

2 charset ASCII

The charset ASCII is defined using the table A.1. The values in the columns headed Character ID and Character Symbol are taken from ASCII (X3.4 ! 1990). The column headed patcode defines which characters match the patcodes A, C, E, L, N, P, and U. The characters in the table with a patcode of A are defined as idents. The collation rule used is *STRING* collation, using the collation order values provided in the table.

Table A.1 - ASCII Character Set Table

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
0	*NUL*	C,E	0		
1	*SOH*	C,E	1		
2	*STX*	C,E	2		
3	*ETX*	C,E	3		
4	*EOT*	C,E	4		
5	*ENQ*	C,E	5		
6	*ACK*	C,E	6		
7	*BELL*	C,E	7		
8	*BS*	C,E	8		
9	*HT*	C,E	9		
10	*LF*	C,E	10		
11	*VT*	C,E	11		
12	*FF*	C,E	12		
13	*CR*	C,E	13		
14	*SO*	C,E	14		
15	*SI*	C,E	15		
16	*DLE*	C,E	16		
17	*DC1*	C,E	17		
18	*DC2*	C,E	18		
19	*DC3*	C,E	19		
20	*DC4*	C,E	20		
21	*NAK*	C,E	21		
22	*SYN*	C,E	22		
23	*ETB*	C,E	23		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
24	*CAN*	C,E	24		
25	*EM*	C,E	25		
26	*SUB*	C,E	26		
27	*ESC*	C,E	27		
28	*FS*	C,E	28		
29	*GS*	C,E	29		
30	*RS*	C,E	30		
31	*US*	C,E	31		
32	SP (space)	P,E	32		
33	!	P,E	33		
34	"	P,E	34		
35	#	P,E	35		
36	$	P,E	36		
37	%	P,E	37		
38	&	P,E	38		
39	' (apostrophe)	P,E	39		
40	(	P,E	40		
41	)	P,E	41		
42	*	P,E	42		
43	+	P,E	43		
44	, (comma)	P,E	44		
45	- (hyphen)	P,E	45		
46	.	P,E	46		
47	/	P,E	47		
48	0	N,E	48		
49	1	N,E	49		
50	2	N,E	50		
51	3	N,E	51		
52	4	N,E	52		
53	5	N,E	53		
54	6	N,E	54		
55	7	N,E	55		
56	8	N,E	56		
57	9	N,E	57		
58	:	P,E	58		
59	;	P,E	59		
60	<	P,E	60		
61	=	P,E	61		
62	>	P,E	62		
63	?	P,E	63		
64	@	P,E	64		
65	A	A,U,E	65		
66	B	A,U,E	66		
67	C	A,U,E	67		
68	D	A,U,E	68		
69	E	A,U,E	69		
70	F	A,U,E	70		
71	G	A,U,E	71		
72	H	A,U,E	72		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
73	I	A,U,E	73		
74	J	A,U,E	74		
75	K	A,U,E	75		
76	L	A,U,E	76		
77	M	A,U,E	77		
78	N	A,U,E	78		
79	O	A,U,E	79		
80	P	A,U,E	80		
81	Q	A,U,E	81		
82	R	A,U,E	82		
83	S	A,U,E	83		
84	T	A,U,E	84		
85	U	A,U,E	85		
86	V	A,U,E	86		
87	W	A,U,E	87		
88	X	A,U,E	88		
89	Y	A,U,E	89		
90	Z	A,U,E	90		
91	[	P,E	91		
92	\	P,E	92		
93	]	P,E	93		
94	^	P,E	94		
95	_ (underscore)	P,E	95		
96	`	P,E	96		
97	a	A,L,E	97		
98	b	A,L,E	98		
99	c	A,L,E	99		
100	d	A,L,E	100		
101	e	A,L,E	101		
102	f	A,L,E	102		
103	g	A,L,E	103		
104	h	A,L,E	104		
105	i	A,L,E	105		
106	j	A,L,E	106		
107	k	A,L,E	107		
108	l	A,L,E	108		
109	m	A,L,E	109		
110	n	A,L,E	110		
111	o	A,L,E	111		
112	p	A,L,E	112		
113	q	A,L,E	113		
114	r	A,L,E	114		
115	s	A,L,E	115		
116	t	A,L,E	116		
117	u	A,L,E	117		
118	v	A,L,E	118		
119	w	A,L,E	119		
120	x	A,L,E	120		
121	y	A,L,E	121		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
122	z	A,L,E	122		
123	{	P,E	123		
124	|	P,E	124		
125	}	P,E	125		
126	~	P,E	126		
127	*DEL*	C,E	127		

Note: 2nd and 3rd order collation values happen to be blank (i.e., not needed) for this Character Set Profile definition; the 1st order collation value happens to be unique across all the characters in this profile.

3 charset JIS90

The charset JIS90 supports an encoding of Japanese characters. The specification for this was developed by the MUMPS Development Coordinating Committee - Japan and is described in JIS X0201!1990 and JIS X0208!1990. The English translation is partially reproduced in Annex G for information purposes. The reader should refer to JIS X0201!1990 and JIS X0208!1990 for full definition.

(Note that Annex G is informational.)

4 charset ISO-8859-USA

The charset ISO-8859!1-USA is defined using the table A.2. The values in the columns headed Character ID and Character Symbol are taken from ISO-8859!1 (ISO Latin 1). The column headed patcode defines which characters match the patcodes A, C, E, I, L, N, P, and U. The characters in the table with a patcode of A are defined as idents. The collation rule used is *STRING* collation, using the collation order values provided in the table: note that all collation is left-to-right precedence. Note also that the patcode I matches any non-ASCII characters (Character ID number greater than 127), not just those listed in this charset.

5 charset ISO-8859!1-USA/M

The charset ISO-8859!1-USA/M is defined using the table A.2. The values in the columns headed Character ID and Character Symbol are taken from ISO-8859!1 (ISO Latin 1). The column headed patcode defines which characters match the patcodes A, C, E, I L, N, P, and U. The characters in the table with a patcode of A are defined as idents. The collation rule used is *M[UMPS]* collation, using the collation order values provided in the table: note that all collation is left-to-right precedence. Note also that the patcode I matches any non-ASCII characters (Character ID number greater than 127), not just those listed in this charset.

Table A.2 - ISO-8859!1-USA Character Set Table

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
0	*NUL*	C,E	0		
1	*SOH*	C,E	1		
2	*STX*	C,E	2		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
3	*ETX*	C,E	3		
4	*EOT*	C,E	4		
5	*ENQ*	C,E	5		
6	*ACK*	C,E	6		
7	*BELL*	C,E	7		
8	*BS*	C,E	8		
9	*HT*	C,E	9		
10	*LF*	C,E	10		
11	*VT*	C,E	11		
12	*FF*	C,E	12		
13	*CR*	C,E	13		
14	*SO*	C,E	14		
15	*SI*	C,E	15		
16	*DLE*	C,E	16		
17	*DC1*	C,E	17		
18	*DC2*	C,E	18		
19	*DC3*	C,E	19		
20	*DC4*	C,E	20		
21	*NAK*	C,E	21		
22	*SYN*	C,E	22		
23	*ETB*	C,E	23		
24	*CAN*	C,E	24		
25	*EM*	C,E	25		
26	*SUB*	C,E	26		
27	*ESC*	C,E	27		
28	*FS*	C,E	28		
29	*GS*	C,E	29		
30	*RS*	C,E	30		
31	*US*	C,E	31		
32	SP (space)	P,E	32		
33	!	P,E	33		
34	"	P,E	34		
35	#	P,E	35		
36	$	P,E	36		
37	%	P,E	37		
38	&	P,E	38		
39	' (apostrophe)	P,E	39		
40	(	P,E	40		
41	)	P,E	41		
42	*	P,E	42		
43	+	P,E	43		
44	, (comma)	P,E	44		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
45	- (hyphen)	P,E	45		
46	.	P,E	46		
47	/	P,E	47		
48	0	N,E	48		
49	1	N,E	49		
50	2	N,E	50		
51	3	N,E	51		
52	4	N,E	52		
53	5	N,E	53		
54	6	N,E	54		
55	7	N,E	55		
56	8	N,E	56		
57	9	N,E	57		
58	:	P,E	58		
59	;	P,E	59		
60	<	P,E	60		
61	=	P,E	61		
62	>	P,E	62		
63	?	P,E	63		
64	@	P,E	64		
65	A	A,U,E	65	1	1
66	B	A,U,E	66	1	1
67	C	A,U,E	67	1	1
68	D	A,U,E	68	1	1
69	E	A,U,E	70	1	1
70	F	A,U,E	71	1	1
71	G	A,U,E	72	1	1
72	H	A,U,E	73	1	1
73	I	A,U,E	74	1	1
74	J	A,U,E	75	1	1
75	K	A,U,E	76	1	1
76	L	A,U,E	77	1	1
77	M	A,U,E	78	1	1
78	N	A,U,E	79	1	1
79	O	A,U,E	80	1	1
80	P	A,U,E	81	1	1
81	Q	A,U,E	82	1	1
82	R	A,U,E	83	1	1
83	S	A,U,E	84	1	1
84	T	A,U,E	85	1	1
85	U	A,U,E	86	1	1
86	V	A,U,E	87	1	1

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
87	W	A,U,E	88	1	1
88	X	A,U,E	89	1	1
89	Y	A,U,E	90	1	1
90	Z	A,U,E	91	1	1
91	[	P,E	93		
92	\	P,E	94		
93	]	P,E	95		
94	^	P,E	96		
95	_ (underscore)	P,E	97		
96	`	P,E	98		
97	a	A,L,E	65	0	1
98	b	A,L,E	66	0	1
99	c	A,L,E	67	0	1
100	d	A,L,E	68	0	1
101	e	A,L,E	70	0	1
102	f	A,L,E	71	0	1
103	g	A,L,E	72	0	1
104	h	A,L,E	73	0	1
105	i	A,L,E	74	0	1
106	j	A,L,E	75	0	1
107	k	A,L,E	76	0	1
108	l	A,L,E	77	0	1
109	m	A,L,E	78	0	1
110	n	A,L,E	79	0	1
111	o	A,L,E	80	0	1
112	p	A,L,E	81	0	1
113	q	A,L,E	82	0	1
114	r	A,L,E	83	0	1
115	s	A,L,E	84	0	1
116	t	A,L,E	85	0	1
117	u	A,L,E	86	0	1
118	v	A,L,E	87	0	1
119	w	A,L,E	88	0	1
120	x	A,L,E	89	0	1
121	y	A,L,E	90	0	1
122	z	A,L,E	91	0	1
123	{	P,E	99		
124	|	P,E	100		
125	}	P,E	101		
126	~	P,E	102		
127	*DEL*	C,E	103		
128		C,E,I	104		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
129		C,E,I	105		
130		C,E,I	106		
131		C,E,I	107		
132	*IND*	C,E,I	108		
133	*NEL*	C,E,I	109		
134	*SSA*	C,E,I	110		
135	*HTS*	C,E,I	111		
136	*HTJ*	C,E,I	112		
137	*VTS*	C,E,I	113		
138	*PLD*	C,E,I	114		
139	*PLU*	C,E,I	115		
140	*RI*	C,E,I	116		
141	*SS2*	C,E,I	117		
142	*SS3*	C,E,I	118		
143	*DCS*	C,E,I	119		
144	*PU1*	C,E,I	120		
145	*PU2*	C,E,I	121		
146	*STS*	C,E,I	122		
147	*CCH*	C,E,I	123		
148	*MW*	C,E,I	124		
149	*SPA*	C,E,I	125		
150	*EPA*	C,E,I	126		
151		C,E,I	127		
152		C,E,I	128		
153		C,E,I	129		
154		C,E,I	130		
155	*CSI*	C,E,I	131		
156	*ST*	C,E,I	132		
157	*OSC*	C,E,I	133		
158	*PM*	C,E,I	134		
159	*APC*	C,E,I	135		
160	*NBSP*	C,E,I	136		
161	¡	P,E,I	137		
162	¢	P,E,I	138		
163	£	P,E,I	139		
164	¤	P,E,I	140		
165	¥	P,E,I	141		
166	*	P,E,I	142		
167	§	P,E,I	143		
168	(	P,E,I	144		
169	©	P,E,I	145		
170	ª	P,E,I	146		

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
171	«	P,E,I	147		
172	¬	P,E,I	148		
173	—	P,E,I	149		
174	®	P,E,I	150		
175	–	P,E,I	151		
176	0	P,E,I	152		
177	±	P,E,I	153		
178	2	P,E,I	154		
179	3	P,E,I	155		
180	5	P,E,I	156		
181	µ	P,E,I	157		
182	¶	P,E,I	158		
183	"	P,E,I	159		
184	2	P,E,I	160		
185	1	P,E,I	161		
186	/	P,E,I	162		
187	»	P,E,I	163		
188	¼	P,E,I	164		
189	½	P,E,I	165		
190	¾	P,E,I	166		
191	¿	P,E,I	167		
192	À	A,U,E,I	65	1	3
193	Á	A,U,E,I	65	1	2
194	Â	A,U,E,I	65	1	4
195	Ã	A,U,E,I	65	1	6
196	Ä	A,U,E,I	65	1	5
197	Å	A,U,E,I	65	1	10
198	Æ	A,U,E,I	65 70	1 1	1 0
199	Ç	A,U,E,I	67	1	13
200	È	A,U,E,I	70	1	3
201	É	A,U,E,I	70	1	2
202	Ê	A,U,E,I	70	1	4
203	Ë	A,U,E,I	70	1	5
204	Ì	A,U,E,I	74	1	3
205	Í	A,U,E,I	74	1	2
206	Î	A,U,E,I	74	1	4
207	Ï	A,U,E,I	74	1	5
208	o	A,U,E,I	69	1	1
209	Ñ	A,U,E,I	79	1	6
210	Ò	A,U,E,I	80	1	3
211	Ó	A,U,E,I	80	1	2

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
212	Ô	A,U,E,I	80	1	4
213	Õ	A,U,E,I	80	1	6
214	Ö	A,U,E,I	80	1	5
215	×	P,E,I	168		
216	Ø	A,U,E,I	80	1	16
217	Ù	A,U,E,I	86	1	3
218	Ú	A,U,E,I	86	1	2
219	Û	A,U,E,I	86	1	4
220	Ü	A,U,E,I	86	1	5
221	Ý	A,U,E,I	90	1	2
222	Þ	A,U,E,I	92	1	1
223	ß	A,L,E,I	84 84	0 0	1 0
224	à	A,L,E,I	65	0	3
225	á	A,L,E,I	65	0	2
226	â	A,L,E,I	65	0	4
227	ã	A,L,E,I	65	0	6
228	ä	A,L,E,I	65	0	5
229	å	A,L,E,I	65	0	10
230	æ	A,L,E,I	65 70	0 0	1 0
231	ç	A,L,E,I	67	0	13
232	è	A,L,E,I	70	0	3
233	é	A,L,E,I	70	0	2
234	ê	A,L,E,I	70	0	4
235	ë	A,L,E,I	70	0	5
236	ì	A,L,E,I	74	0	3
237	í	A,L,E,I	74	0	2
238	î	A,L,E,I	74	0	4
239	ï	A,L,E,I	74	0	5
240	ð	A,L,E,I	69	0	1
241	ñ	A,L,E,I	79	0	6
242	ò	A,L,E,I	80	0	3
243	ó	A,L,E,I	80	0	2
244	ô	A,L,E,I	80	0	4
245	õ	A,L,E,I	80	0	6
246	ö	A,L,E,I	80	0	5
247	÷	P,E,I	169		
248	ø	A,L,E,I	80	0	16
249	ù	A,L,E,I	86	0	3
250	ú	A,L,E,I	86	0	2
251	û	A,L,E,I	86	0	4
252	ü	A,L,E,I	86	0	5

Character ID	Character Symbol	patcode	Collation Table		
			1st Order	2nd Order	3rd Order
253	ý	A,L,E,I	90	0	2
254	þ	A,L,E,I	92	0	1
255	ÿ	A,L,E,I	90	0	5

Note: unique collation requires that no two rows of this table have identical collation order columns.

Editor's note:
I know that this was discussed before, but it still strikes me as "astonishing" that character codes 178, 179, 185, 188, 189 and 190 do not match the pattern code N.

Editor's note:
Character code 166 is supposed to be a vertical bar with a slit in the middle. I'll have to find a way to make my text processor produce that character.

Annex B: Error code translations (informative)

M1 Naked indicator undefined
M2 Invalid combination with P fncodatom
M3 $RANDOM seed less than 1
M4 No true condition in $SELECT
M5 lineref less than zero
M6 Undefined lvn
M7 Undefined gvn
M8 Undefined svn
M9 Attempt to divide by zero
M10 Invalid pattern match range
M11 No parameters passed
M12 Invalid lineref (negative offset)
M13 Invalid lineref (line not found)
M14 line level not 1
M15 Undefined index variable
M16 Argumented QUIT not allowed
M17 Argumented QUIT required
M18 Fixed length READ not greater than zero
M19 Cannot copy a tree or subtree into itself
M20 line must have formallist
M21 Algorithm specification invalid

Editor's note:
The actual meaning of error M21 is: Multiple formal parameters with the same name.
Suggest to modify the description accordingly.

M22 SET or KILL to ^$GLOBAL when data in global variable
M23 SET or KILL to ^$JOB for non-existent job number
M24 Change to collation algorithm while subscripted local variables defined
M25 RSAVE to currently executing routine
M26 Non-existent environment
M27 Attempt to rollback a transaction that is not restartable
M28 Mathematical function, parameter out of range
M29 SET or KILL on ssvn not allowed by implementation
M30 Reference to glvn with different collating sequence within a collating algorithm
M31 controlmnemonic used for device without a mnemonicspace selected
M32 controlmnemonic used in user-defined mnemonicspace that has no associated line
M33 SET or KILL to ^$ROUTINE when routine exists
M34 This error code is not currently assigned
M35 Device does not support mnemonicspace
M36 Incompatible mnemonicspaces
M37 READ from device identified by the empty string
M38 Invalid ssvn subscript
M39 Invalid $NAME argument

Editor's note:
The actual meaning of error M39 is: Name of variable expected.
Suggest to modify the description accordingly.

M40 Call-by-reference in JOB actual
M41 Invalid LOCK argument within a TRANSACTION
M42 Invalid QUIT within a TRANSACTION
M43 Invalid range value ($X,$Y)
M44 Invalid command outside of a TRANSACTION
M45 Invalid GOTO reference
M46 *This error code is assigned to X11.6, MWAPI*
M47 Invalid attribute value

M48 *This error code is assigned to X11.6, MWAPI*
M49 *This error code is assigned to X11.6, MWAPI*
M50 *This error code is assigned to X11.6, MWAPI*
M51 *This error code is assigned to X11.6, MWAPI*
M52 *This error code is assigned to X11.6, MWAPI*
M53 *This error code is assigned to X11.6, MWAPI*
M54 *This error code is assigned to X11.6, MWAPI*
M55 *This error code is assigned to X11.6, MWAPI*

Editor's note:
Error codes M46 through M55 are part of MWAPI. M47 is also used in the language standard, hence its description appears in this document.
Suggest to include all descriptions from M46 through M55 here.
M46: Invalid attribute name
M47: Invalid attribute name
M48: Nonexistent window, element or choice
M49: Invalid attempt to set focus
M50: Attempt to reference a non M-Term window in an OPEN command
M51: Attempt to destroy M-Term window prior to CLOSE
M52: Required attribute missing
M53: Invalid argument for font function
M54: Attempt to create non-modal child of a modal parent
M55: Invalid nested ESTART command
Further suggest to rename M46 and M47 as:
M46: Attempt to assign value to reserved attribute
or: Name of attribute not valid in current context
M47: Cannot modifiy TIED attribute
or: Value for attribute not valid in current context

M56 name length limit exceeded
M57 More than one defining occurence of label in routine
M58 Too few formal parameters
M59 Environment references not permitted for this ssvn
M60 Reference to undefined ssvn with unspecified semantics
M61 This error code is not currently assigned
M62 This error code is not currently assigned
M63 This error code is not currently assigned
M64 This error code is not currently assigned
M65 This error code is not currently assigned
M66 This error code is not currently assigned
M67 This error code is not currently assigned
M68 This error code is not currently assigned
M69 This error code is not currently assigned
M70 This error code is not currently assigned
M71 This error code is not currently assigned
M72 This error code is not currently assigned
M73 This error code is not currently assigned
M74 This error code is not currently assigned
M75 String length limit exceeded
M88 RLOAD from a non-existent routine
M90 Invalid namevalue
M92 Arithmetic overflow
M94 Attempt to raise 0 to the power of 0
M95 Result-value has non-zero imaginary part
M96 SET or KILL to write-once ssvn node when $DATA equals 1 or 11
M97 Routine for user-defined ssvn not found
M98 Resource unavailable
M99 Invalid operation for socket context
M100 Output timeout expired

M101 Attempt to assign incorrect value to $ECODE
M102 Events cannot be both synchronous and asynchronous
M103 Invalid event identifier
M104 IPC event identifier is not a valid job-number
M105 Object not currently accessible
M106 Object does not support requested method or property
M107 Object has no default value
M108 Value is not of data type OREF
M109 Undefined devicekeyword
M110 Event identifier not available
M111 Invalid number of days for date
M112 Invalid number of seconds for time
M113 Invalid separator inserted

S0 Invalid syntax

Annex C: Metalanguage element dictionary (Informative)

::=	definition
[ ]	optional element
| |	group of alternate choices
...	optional indefinite repetition
ablockargument	asynchronous event argument
actual	actual parameter
actualkeyword	actual parameter keyword
actuallist	actual parameter list
actualname	actual parameter name
algoref	algorithm reference
alternation	alternation
argument	argument of a command
assignargument	ASSIGN argument
assigndestination	ASSIGN destination
assignleft	ASSIGN left
binaryop	binary operator
CB	close bracket character
charset	character set
charsetexpr	character set expression
charspec	character specification
closeargument	CLOSE argument
command	command
commands	commands separated by cs
commandword	command word
comment	comment
controlmnemonic	control mnemonic
CR	carriage return character
cs	command separator
descriptor	value for character set profile
descsep	separator in name of character set profile
device	device
deviceattribute	device attribute
devicecommand	device command
devicekeyword	device keyword
deviceparam	device parameter
deviceparameters	device parameters
devicexpr	device expression
devn	implementation-specific identifier for a device
digit	decimal digit
dlabel	indirect label (evaluated label)
doargument	DO argument
ecode	error code
einforef	event information reference
einfoattribute	event information attribute, see X11.6
<embedded SQL declare section>	SQL declarative statement(s)
<embedded SQL MUMPS program>	SQL program, embedded in a M[UMPS] program
<embedded SQL statement>	single SQL statement
emptystring	empty string
entryref	entry reference
environment	set of distinct names
eoffset	error offset
eol	end-of-line
eor	end-of-routine
erchar	error character
erspec	event restricted specifier
espec	event specifier, see X11.6
especref	event specifier reference
estartargument	ESTART argument
evid	event id
evclass	event class
eventexpr	event expression
exfunc	extrinsic function
exp	exponent
expr	expression
expratom	expression atom
expritem	expression item
exprtail	expression tail
externalroutinename	external routine name
externref	external reference
extid	external identifier
extractfields	fields in multi-part record
extracttemplate	template for multi-part record
extsyntax	external syntax
exttext	external text
exvar	extrinsic variable
fchar	single character value
fdirectives	directives for localized formatting
FF	form feed character
ffformat	form feed format code
fieldindex	index for field in multi-part record
fieldwidth	width of field in multi-part record
fmask	mask for localized formatting
fncodatom	$FNUMBER code atom
fncode	$FNUMBER code
fncodexpr	$FNUMBER code expression
fncodp	$FNUMBER code P
fncodt	$FNUMBER code T
formalline	formal line (line with formallist)
formallist	formal argument list
format	I/O format code
forparameter	FOR range specification
fservice	service name with parameters
function	intrinsic function
functionname	function name
glvn	global or local variable name
gnamind	global variable name

	indirection
gotoargument	GOTO argument
graphic	graphic (character with visible representation)
gvn	global variable name
gvnexpr	global variable name expression
hangargument	HANG argument
ident	identification
ifargument	IF argument
initialrecordvalue	initial value for multi-part record processing
intexpr	expr, value interpreted as an integer
intlit	integer literal
iocommand	I/O command
jobargument	JOB argument
jobenv	JOB environment
jobparameters	JOB parameters
killarglist	KILL argument list
killargument	KILL argument
L	list (list of)
label	label of a line
labelref	label reference
leftexpr	left expression
leftrestricted	left restricted
levelline	level line (line without formallist)
LF	line feed character
li	level indicator
libdatatype	library data type
library	library
libraryelement	library element
libraryelementdef	library element definition
libraryelementexpr	library element expression
libraryexpr	library expression
libraryopt	library optional flag
libraryparam	library element parameter
libraryref	library element reference
libraryresult	library element result
line	line in routine
linebody	line body
lineref	line reference
lname	local name
lnamind	local name indirection
lockargument	LOCK argument
logicalop	logical operator
ls	label separator
lvn	local variable name
lvnexpr	name of local variable
mant	mantissa
mergeargument	MERGE argument
mnemonicspace	mnemonic space
mnemonicspacename	mnemonic space name
mnemonicspec	mnemonic space specifier
<MUMPS character variable>	variable length string
<MUMPS host identifier>	M[UMPS] variable name
<MUMPS length specification>	length of string
<MUMPS numeric variable>	variable with numeric value
mumpsreturn	$MUMPS return
<MUMPS variable definition>	definition of one or more M[UMPS] variables
mval	M value (string)
name	name
namedactual	named actual parameter
namedactuallist	named actual parameter list
namevalue	name value
newargument	NEW argument
newsvn	NEW svn
nlformat	new line format code
noncomma	non-comma
noncommasemi	non-comma or -semicolon
nonquote	non-quote (any graphic not equal to quote)
nref	name reference
numexpr	expression, value interpreted numerically
numlit	numeric literal
OB	open bracket character
object	expression atom, value interpreted as an OREF
openargument	OPEN argument
openparameters	OPEN parameters
oref	object reference value
owmethod	object with method
owproperty	object with property
owservice	object with service
packagename	package name
patatom	pattern atom
patcode	pattern code
patgrp	pattern atom group
patnonY	pattern non Y
patnonYZ	pattern non Y or Z
patnonZ	pattern non Z
patsetdest	pattern SET destination
patstr	pattern string literal
pattern	pattern
piecedelimiter	delimiter for multi-part record
place	place
positionformat	position format code
postcond	post condition
<precision>	precision of value of SQL variable
processid	process identifier
processparameters	process parameters
readargument	READ argument
readcount	READ count
recordfieldglvn	variable holding mult-part record
recordfieldvalue	value for field in multi-part

	record
relation	relational operator
repcount	repeat count in patatom
restartargument	restart argument
rexpratom	restricted expression atom
rgvn	restricted global variable name
rlvn	restricted local variable name
rnref	restricted name reference
routine	routine
routineargument	routine load or save argument
routineattribute	routine attribute
routinebody	routine body
routinehead	routine head
routinekeyword	routine keyword
routinename	routine name
routineparam	routine parameter
routineparameters	routine parameters
routineref	routine reference
routinexpr	routine expression
rssvn	restricted structured system variable name
<scale>	order of magnitude of value of SQL variable
servicename	service name
setargument	SET argument
setdestination	SET destination
setdextract	SET destination for field in multi-part record
setdpiece	SET destination for field in multi-part record
setev	SET error variable
setextract	SET $EXTRACT
setleft	SET left
setpiece	SET $PIECE
setqsub	SET $QSUBSCRIPT
SP	space character
ssvn	structured system variable name
ssvname	structured system variable name
ssvnamind	structured system variable name indirection
stackcode	$STACK code
stackcodexpr	$STACK code expression
strconst	string constant
strlit	string literal
sublit	subscript literal
subnonquote	subscript non-quote
svn	special variable name
system	system
systemexpr	system expression
tabformat	tab format code
textarg	$TEXT argument
timeout	time-out specification
transparameters	transaction parameters
truthop	truth operator
tsparam	TSTART parameter
tstartargument	TSTART argument
tstartkeyword	TSTART keyword
tvexpr	expr, value interpreted as a truth-value
unaryop	unary operator
useargument	USE argument
V	value (evaluates to)
VB	vertical bar character
wevclass	windows event class
writeargument	WRITE argument
xargument	EXECUTE argument

Annex D: Embedded SQL (Informative)

SQL2 provides a capability for supporting embedded SQL MUMPS programs. The specification for this is described in ANSI X3.135 (ISO/IEC 9075, 1992) and is partially reproduced here for information purposes. The reader should refer to ANSI X3.135 Section 19 Embedded SQL for the full definition.

"19.1 <embedded SQL host program>

...

Syntax Rules

1. An <embedded SQL host program> is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this standard.

2. An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is "<ampersand>SQL<open paren>". There shall be no <separator> between the <ampersand> and "SQL" nor between "SQL" and the <open paren>.

...

3. **...**
   An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL terminator> that is a <close paren>.

4. The <token>s comprising an <SQL prefix>, <embedded SQL begin declare>, or <embedded SQL end declare> shall be separated by <space> characters and be specified on one line. Otherwise, the rules for the continuation of lines and tokens from one line to the next and for the placement of host language comments are those of the programming language of the containing <embedded SQL host program>.

...

19.7 <embedded SQL MUMPS program>

Function

Specify an <embedded SQL MUMPS program>

Format

<embedded SQL declare section> ::= !! *See the Syntax Rules.*

<embedded SQL MUMPS program> ::= !! *See the Syntax Rules.*

<embedded SQL statement> ::= !! *See the Syntax Rules.*

<MUMPS host identifier> ::= !! *See the Syntax Rules.*

<MUMPS variable definition> ::=

{ <MUMPS numeric variable> | <MUMPS character variable> } <semicolon>

<MUMPS character variable> ::=
VARCHAR <MUMPS host identifier> <MUMPS length specification>
[{ , <MUMPS host identifier> <MUMPS length specification> }...]

<MUMPS length specification> ::=
<open paren> <length> <close paren>

<MUMPS numeric variable> ::=
{ INT
| DEC [{ <precision> [, <scale>])]
| REAL }
<MUMPS host identifier> [{ , <MUMPS host identifier> }...]

<precision> ::= !! *See the Syntax Rules.*

<scale> ::= !! *See the Syntax Rules.*

Syntax Rules

1. An <embedded SQL MUMPS program> is a compilation unit that consists of MUMPS text and SQL text. The MUMPS text shall conform to standard MUMPS. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2. A <MUMPS host identifier> is any valid MUMPS variable name. A <MUMPS host identifier> shall be contained in an <embedded SQL MUMPS program>.

3. An <embedded SQL statement> may be specified wherever a MUMPS command may be specified.

4. A <MUMPS variable definition> defines one or more host variables.

5. The <MUMPS character variable> defines a variable-length string. The equivalent SQL data type is VARCHAR whose maximum length is the <length> of the <MUMPS length specification>.

6. INT describes an exact numeric variable. The equivalent SQL data type is INTEGER.

7. DEC describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

8. REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

9. An <embedded SQL MUMPS program> shall contain either a variable named SQLCODE defined with a datatype of INT or a variable named SQLSTATE defined with a data type that is VARCHAR with length 5, or both.

   Note: *SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this standard. See Annex D, "Deprecated Features".*

..."

Note that the Annex D mentioned in the last paragraph above refers to the SQL standard, not to the M[UMPS] standard.

Annex E: Transportability of M[UMPS] Software Systems (informative)

The transfer of routines between machine environments is affected by numerous machine and operating systems factors. A standard transfer format for both routines and data stored within global variables cannot at the same time easily cope with the simple and the complex case efficiently, in addition to dealing with the environmental idiosyncrasies. Therefore, the responsibility for the detailed format is left to the transferor.

1 Routine Transfer Format

The routine loader routine shall have a form that will load the routines from the transfer medium and will save it in internal format. The save routine creating the transfer medium shall produce the following routine transfer format:

```
Header-line!1 eol
Header-line-2 eol
routinehead
routine-line eol
.
.
.
eol
routinehead
routine-line eol
.
.
.
eol
[***RTN END***] eol
```

In the above structure, routine-line is a string in a format as returned by $TEXT. The two header lines shall be free text and may contain any message the sender wishes to convey to the receiver.

NOTE: Each routine is separated by a blank line (an eol) from the following one. Optionally, either two successive blank lines or the string "***RTN END***" denotes the end of the file. Eol is defined to be a logical end-of-line record as mutually defined by the sending and receiving environments.

2 Global Variable Transfer Format

The global variable loader shall read and the global variable saver shall produce on the transfer medium the following transfer format:

```
Header-line!1 eol
Header-line-2 eol
Full global variable reference eol
Data contents eol
.
.
.
Full global variable reference eol
Data contents eol
eol
```

`[***GBL END***] eol`

Eol is defined to be a logical end-of-line record as mutually defined by the sending and receiving environments.

The full global variable reference shall conform to a global variable name specification as defined by ANSI/MDC X11.1!1994 section 1, subclause 7.1.2.4. When data contains ASCII control characters, decimal (0-31,127), the user shall be responsible for handling the accurate reconstruction of the data string in the host environment. Subscripts in the full global variable reference shall not contain the ASCII control characters decimal 0-31 or 127. Optionally, either two successive blank lines or the string "***GBL END***" denotes the end of the file.

Annex F: X3.64 ControlmnemonicS (informative)

control-mnemonic	Control Function
APC	Application Program Command
CBT	Cursor Backward Tabulation
CCH	Cancel Character
CHA	Cursor Horizontal Absolute
CHT	Cursor Horizontal Tabulation
CNL	Cursor Next Line
CPL	Cursor Preceding Line
CPR	Cursor Position Report
CTC	Cursor Tabulation Control
CUB	Cursor Backward
CUD	Cursor Down
CUF	Cursor Forward
CUP	Cursor Position
CUU	Cursor Up
CVT	Cursor Vertical Tabulation
DA	Device Attributes
DAQ	Define Area Qualification
DCH	Delete Character
DCS	Device Control String
DL	Delete Line
DMI	Disable Manual Input
DSR	Device Status Report
EA	Erase in Area
ECH	Erase Character
ED	Erase in Display
EF	Erase in Field
EL	Erase in Line
EMI	Enable Manual Input
EPA	End of Protected Area
ESA	End of Selected Area
FNT	Font Selection
GSM	Graphic Size Modification
GSS	Graphic Size Selection
HPA	Horizontal Position Absolute
HPR	Horizontal Position Relative
HTJ	Horizontal Tab with Justify
HTS	Horizontal Tabulation Set
HVP	Horizontal and Vertical Position
ICH	Insert Character
IL	Insert Line
IND	Index
INT	Interrupt
JFY	Justify
MC	Media Copy
MW	Message Waiting
NEL	Next Line
NP	Next Page
OSC	Operating System Command
PLD	Partial Line Down
PLU	Partial Line Up
PM	Privacy Message
PP	Preceding Page
PU1	Private Use One
PU2	Private Use Two
QUAD	QUAD
REP	Repeat
RI	Reverse Index
RIS	Reset to Initial State
RM	Reset Mode
SD	Scroll Down
SEM	Select Editing Extent Mode
SGR	Select Graphic Rendition
SL	Scroll Left
SM	Set Mode
SPA	Start of Protected Area
SPI	Spacing Increment
SR	Scroll Right
SS2	Single Shift Two
SS3	Single Shift Three
SSA	Start of Selected Area
ST	String Terminator
STS	Set Transmit State
SU	Scroll Up
TBC	Tabulation Clear
TSS	Thin Space Specification
VPA	Vertical Position Absolute
VPR	Vertical Position Relative
VTS	Vertical Tabulation Set

Annex G: charset JIS90 (informative)

(This is a partial English reproduction of the JIS90 charset. The reader should refer to JIS X0201!1990 and JIS X0208!1990 for the full definition.)

1 charset JIS90

The charset JIS90 is defined using the JIS X0201!1990 8-bit Code and the JIS X0208!1990 2-Byte code for Information Interchange.

2 JIS X0201!1990

In JIS X0201!1990, the values of decimal and character are the same as those from ASCII (X3.4!1990) in the range between decimal 0!127, except decimal 92 which represents "¥" (yen) instead of "\" and Decimal 126 which represents "‾" (overline) instead of "~" (tilde).

The patcodes defined in charset M as A, C, E, L, N, P, and U apply in the same way in the range of decimal 0!127.

In the decimal range between 161 and 223, the values represent 8-bit katakana characters.

3 JIS X0208!1990

In JIS X0208!1990, the relation of decimal and character is obtained as following. Let C_1 and C_2 be the decimal values of the 1st byte and the 2nd byte code for a character, then the range of decimal code for both C_1 and C_2 is [33,127] and the decimal value of the character is C_1*256+C_2. Let *n* be a decimal value. If there is no character assigned for *n* in JIS X0208, then the external representation of $CHAR(*n*) will be the same as the Japanese space, or $CHAR(8481).

4 Pattern Codes

Patcodes E and 力 (ka, $CHAR(182)) apply for the characters in the decimal range 161-223. Patcodes E and 全 (zen, $CHAR(16692)) apply for the characters in the decimal range 8481-32382.

5 Characters used in names

Characters in the charset JIS90 except $CHAR(8481) may be defined as ident.

6 Collation

The collation scheme of charset JIS90 is ordered by the $ASCII value of the character, whitin each of JIS X0201!1990 and X0208!1990.

Annex H: Sockets Binding (informative)

1 Introduction

Sockets are used to represent and manage a communication channel between two entities on a network. The channel can be connection oriented, in which the two entities establish a session for the duration of the conversation, or it can be connectionless, in which messages are simply sent out to the intended recipient.

2 General

Socket communications are accessed by the use of controlmnemonics and deviceparameters. This binding uses the SOCKET mnemonicspace.

Socket identifiers (simply referred to as "sockets") are used by the implementation to identify the "socket handle" used by the underlying implementation. The actual mapping between socket identifiers and the underlying sockets is implementation specific.

Sockets are accessed and manipulated via a socket device. The socket device can contain a collection of sockets. At any time, a single socket from the collection is the current socket. Any socket in the collection can be designated to be the current socket. Furthermore, sockets can be attached (added) and detached (deleted) from the collection of sockets.

3 Commands and deviceparameters

For the SOCKET mnemonicspace, the following deviceattributes are defined. All deviceattributes and devicekeywords beginning with the letter Z (or z) are reserved for the implementation. All others are reserved. Names differing only in the use of corresponding upper and lower case letters are equivalent.

Once a device is successfully OPENed, the structured system variable ^$DEVICE reflects the current settings of the deviceattributes.

An attempt to modify a socket when none is current will result in an error with ecode = "M99" (invalid operation for socket context). An attempt to specify an invalid argument to a deviceattribute will result in an error with ecode = "M47" (invalid attribute value).

3.1 OPEN and USE Commands

The OPEN and USE commands allow sockets to be associated with devices after specifying a mnemonicspace equal to "SOCKET".

The following deviceattributes are valid on an OPEN or USE command.

3.1.1 ATTACH = expr

expr specifies an implementation-specific socket identifier. It specifies an existing socket that should be added to this device's collection of sockets. If the socket is attached to any other process or device, the ATTACH will fail with ecode = "M98" (resource unavailable). Otherwise, if the operation is successful, the attached socket will become the current socket for the device.

3.1.2 CONNECT = expr

expr specifies implementation-specific context information. A client connection will be established with a server, using the connection information to locate the server. A new socket will be allocated for the client connection and will become the current socket for the device.

3.1.3 DELIMITER = | expr
(L expr) |

expr specifies an I/O delimiter. Each usage of this deviceattribute replaces the existing set of I/O delimiters with a new set (which may be empty). The set is empty if all exprs have the value of the empty string.

If no DELIMITER is specified, the initial set of I/O delimiters for the socket is empty.

3.1.4 IOERROR = expr

expr specifies the I/O error trapping mode.

A value equal to “NOTRAP” specifies that I/O errors on a device do not raise error conditions. A value equal to “TRAP” specifies that I/O errors on a device do raise error conditions with an ecode value associated with the error. Values beginning with Z (or z) are reserved for the implementation. All other values are reserved. Values differing only in the use of corresponding upper and lower case letters are equivalent.

If no IOERROR is specified, the initial I/O error trapping mode for a socket is “NOTRAP”.

3.1.5 LISTEN = expr

expr specifies implementation and protocol specific information. This command causes the device to allocate a new socket and prepare it for listening for incoming requests for connection to a server. The new socket is made the current socket for the device. Requests for connections will not be accepted until a WRITE /controlmnemonic is issued

The following deviceattributes are valid on the USE but not the OPEN command.

3.1.6 DETACH = expr

expr specifies an implementation-specific socket identifier. The specified socket is detached from the device without affecting the sockets existing connection. The socket may then be attached to another socket device using the ATTACH deviceattribute.

3.1.7 SOCKET = expr

expr specifies an implementation-specific socket identifier. The specified socket becomes the current socket.

3.2 CLOSE Command

The following deviceattributes are valid on the CLOSE command.

3.2.1 SOCKET = expr

expr specifies an implementation-specific socket identifier which is the socket associated with the device that is to be closed. If this deviceattribute is specified then any other sockets associated with the device are not closed and the M[UMPS] device is not released.

If the SOCKET deviceattribute is omitted then all sockets associated with the device are closed and the

M[UMPS] device is released.

3.3 READ Command

The READ command may be used to obtain data from a socket.

A READ operation will terminate if any of the following are detected, in the order specified:

- Error condition. $DEVICE reflects the error, $KEY is assigned the empty string. The value returned by the READ command is implementation specific.
- READ timeout. $KEY is assigned the empty string. The READ command returns data received up to the timeout.
- READ delimiter. $KEY is assigned the delimiter string which terminated the READ. The READ command returns data received up to, but not including, the delimiter.
- Fixed-length READ requirements are satisfied. This occurs only after the specified number of characters are received. $KEY is assigned the empty string. The READ command returns the characters received.
- For a stream-oriented protocol, when the buffer is empty the READ waits. When there is at least one character, the READ command returns available characters, up to the maximum string length for the implementation. Note that the number of characters returned is not predictable except to be within the range from one to the maximum string length. $KEY is assigned the empty string.
- For a message-oriented protocol, when a complete message is received, READ returns the message. $KEY is assigned the empty string.

For multi-character I/O delimiters, the possibility exists due to the stream nature of transmissions, that characters which would otherwise match an I/O delimiter may actually be spread across multiple ‘packets’. In the event that the last ‘*n*’ characters received (*n* > 0) match a prefix of one or more I/O delimiters, the implementation must determine if any of the additionally expected characters complete the match with the I/O delimiter(s). One implementation would be to internally issue a timed READ. A timeout of this internally issued timed READ does not affect $TEST. The time associated with this internal timed READ is implementation specific and is not included in the timeout which may have been optionally specified on the actual READ command.

3.4 WRITE Command

The WRITE command may be used to send data to a socket.

Data being transmitted is sent using the urgency mode currently in effect for the socket. The definition and usage of urgency mode is implementation-specific.

WRITE ! appends the first I/O delimiter (see X 3.1.3), if specified, to the internal output buffers for the current device. The process then immediately transfers the internally buffered output data to the underlying binding services. This command does not affect internally buffered input data. $X is set to 0. $Y is incremented by 1.

WRITE # causes the process to immediately transfer any internally buffered output data for the current device to the underlying binding services. No I/O delimiters are implicitly added to the internal output buffer. This command does not affect internally buffered input data. $X and $Y are set to 0.

4 controlmnemonics

controlmnemonic names differing only in the use of corresponding upper and lower case letters are equivalent.

4.1 LISTEN [(expr)]

The use of this controlmnemonic causes the process to establish a queue depth for incoming client connections.

If expr is omitted then the queue depth established will take on an implementation specific value.

4.2 WAIT [(numexpr)]

numexpr is a timeout value.

If the optional numexpr is present, the value must be nonnegative. If it is negative, the value 0 is used. numexpr denotes a *t*-second timeout, where *t* is the value of numexpr. If *t* = 0, the condition is tested. If *t* is positive, execution is suspended until the connection is made, but in any case no longer than *t* seconds.

The use of this controlmnemonic causes the process to wait for an event to occur on any socket associated with the device, subject to timeout. When this operation completes, $KEY contains a value identifying the event that occurred.

In the event of a timeout or an error the empty string is returned in $KEY.

If a listening server socket receives a connection request, $KEY will contain the value “CONNECT”. A new socket will be allocated to handle the connection with the client, and the new socket will become the current socket of the device.

If a message is received by a connectionless protocol, $KEY will contain the value “READ”. The socket which received the message will become the current socket of the device.

5 ^$DEVICE

The following nodes are defined in ^$DEVICE for the SOCKET mnemonicspace:

^$DEVICE(device,“SOCKET”) = intexpr
Each device has a collection of sockets associated with it. Each new socket is identified by a socket identifier which is assigned an index number in the collection of sockets. This node of ^$DEVICE defines the index number of the current socket.

^$DEVICE(device,“SOCKET”,index,“DELIMITER”) = intexpr
This provides the number of I/O delimiters, as defined using the DELIMITER deviceattribute, in effect for the device/socket. (See 3.1.3)

^$DEVICE(device,“SOCKET”,index,“DELIMITER”,*n*) = expr
This provides the *n*-th I/O delimiter string. (See 3.1.3)

^$DEVICE(device,“SOCKET”,index,“IOERROR”) = expr
I/O error trapping mode. (See 3.1.4)

^$DEVICE(device,“SOCKET”,index,“LOCALADDRESS”) = expr
This provides the local network node address of the connection

^$DEVICE(device,“SOCKET”,index,“PROTOCOL”) = expr
This provides the network protocol used for the connection

^$DEVICE(device,“SOCKET”,index,“REMOTEADDRESS”) = expr
This provides the remote network node address of the connection

^$DEVICE(device,“SOCKET”,index,“SOCKETHANDLE”) = expr

The value of this node is an implementation-specific string that provides the socket identifier of the indicated socket.

Annex I: Example Code for Library Functions (informative)

The code in this annex is an example of a possible implementation of these library functions. Implementors are encouraged to provide implementations that offer a better efficiency as well as greater accuracy.

1 CHARACTER Library

1.1 COLLATE

```
COLLATE(A,CHARMOD) New x
      Set x=“”
      If $Get(CHARMOD)’=“” Do
      . If $Extract(CHARMOD,1)=“^” Do
      . . Set x=$Extract(CHARMOD,2,$Length(CHARMOD))
      . . If x’=“” Set x=$Get(^$Global(x,“COLLATE”))
      . . Quit
      . If x=“” Set x=$Get(^$Character(CHARMOD,“COLLATE”))
      . Quit
      If x=“” Set x=^$Job($Job,“COLLATE”)
      Set x=@(x_“(”_A_“)”)
      Quit x
```

1.2 COMPARE

```
COMPARE(A,B,CHARMOD) New x,y
      ; Assume current collation, i.e. ]], if no CHARMOD specified
      If $Get(CHARMOD)=“” Quit $Select(A=B:0,A]]B:1,1:!1)
      ;
      ; Otherwise need to override and do string compare
      ; on collation value
      Set x=$%COLLATE^CHARACTER(A,CHARMOD)
      Set y=$%COLLATE^CHARACTER(B,CHARMOD)
      Quit $Select(x=y:0,x]y:1,1:!1)
```

1.3 LOWER

Editor’s note:
Recommend to use $QSUBSCRIPT to find the value of y.
The first subscript of ^$GLOBAL must be a string that starts with “^”, and that character is explicitly removed from the value that is being used.

```
LOWER(A,CHARMOD) New lo,up,x,y
      ;
      ; The code below was approved in document X11/1998-21
      ;
      Set x=$Get(CHARMOD)
      Set lo="abcdefghijklmnopqrstuvwxyz"
      Set up="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      If x?1"^"1E.E Do
      . Set x=$Extract(x,2,$Length(x))
      . If x?1"|".E Do
      . . Set x=$REverse($Extract(x,2,$Length(x)))
      . . Set y=$REverse($PIECE(x,"|",2,$Length(x)+2))
      . . Set x=$REverse($PIECE(x,"|",1))
      . . Set x=$Get(^|y|$GLOBAL(x,"CHARACTER"))
      . . Quit
```

```
      . Else  Set x=$Get(^$GLOBAL(x,"CHARACTER"))
      . Quit
      If x="" Set x=^$JOB($JOB,"CHARACTER")
      Set x=$Get(^$CHARACTER(x,"LOWER"))
      If x="" Quit $TRanslate(A,up,lo)
      Set @("x="_x_"(A)")
      Quit x
```

1.4 PATCODE

Editor's note:
Recommend to use $QSUBSCRIPT to find the value of y.
The first subscript of ^$GLOBAL must be a string that starts with "^", and that character is explicitly removed from the value that is being used.

```
PATCODE(A,PAT,CHARMOD) New x,y
      ;
      ; The code below was approved in document X11/1998-21
      ;
      Set x=$Get(CHARMOD)
      If x?1"^"1E.E Do
      . Set x=$Extract(x,2,$Length(x))
      . If x?1"|".E Do
      . . Set x=$REverse($Extract(x,2,$Length(x)))
      . . Set y=$REverse($PIECE(x,"|",2,$Length(x)+2))
      . . Set x=$REverse($PIECE(x,"|",1))
      . . Set x=$Get(^|y|$GLOBAL(x,"CHARACTER"))
      . . Quit
      . Else  Set x=$Get(^$GLOBAL(x,"CHARACTER"))
      . Quit
      If x="" Set x=^$JOB($JOB,"CHARACTER")
      Set x=$Get(^$CHARACTER(x,"PATCODE",PAT))
      If x="" Quit 0
      Set @("x="_x_"(A)")
      Quit x
```

1.5 UPPER

Editor's note:
Recommend to use $QSUBSCRIPT to find the value of y.
The first subscript of ^$GLOBAL must be a string that starts with "^", and that character is explicitly removed from the value that is being used.

```
UPPER(A,CHARMOD) New lo,up,x,y
      ;
      ; The code below was approved in document X11/1998-21
      ;
      Set x=$Get(CHARMOD)
      Set lo="abcdefghijklmnopqrstuvwxyz"
      Set up="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      If x?1"^"1E.E Do
      . Set x=$Extract(x,2,$Length(x))
      . If x?1"|".E Do
      . . Set x=$REverse($Extract(x,2,$Length(x)))
      . . Set y=$REverse($PIECE(x,"|",2,$Length(x)+2))
      . . Set x=$REverse($PIECE(x,"|",1))
      . . Set x=$Get(^|y|$GLOBAL(x,"CHARACTER"))
      . . Quit
      . Else  Set x=$Get(^$GLOBAL(x,"CHARACTER"))
      . Quit
```

```
      If x="" Set x=^$JOB($JOB,"CHARACTER")
      Set x=$Get(^$CHARACTER(x,"UPPER"))
      If x="" Quit $TRanslate(A,lo,up)
      Set @("x="_x_"(A)")
      Quit x
```

2 MATH Library

2.1 ABS

```
ABS(X) Quit $Translate(+X,"-")
```

2.2 ARCCOS

Option 1, optimized for speed, not precision.

```
ARCCOS(X) ;
      ; This version of the function is
      ; optimized for speed, not for precision.
      ; The 'precision' parameter is not supported,
      ; and the precision is at best 2 in 10**-8.
      ;
      New A,N,R,SIGN,XX
      If X<-1 Set $Ecode=",M28,"
      If X>1 Set $Ecode=",M28,"
      Set SIGN=1 Set:X<0 X=-X,SIGN=-1
      Set A(0)=1.5707963050,A(1)=-0.2145988016,A(2)=0.0889789874
      Set A(3)=-0.0501743046,A(4)=0.0308918810,A(5)=-0.0170881256
      Set A(6)=0.0066700901,A(7)=-0.0012624911
      Set R=A(0),XX=1 For N=1:1:7 Set XX=XX*X,R=A(N)*XX+R
      Set R=$%SQRT^MATH(1-X,11)*R
      ;
      Quit R*SIGN
```

Option 2, optimized for precision, not speed.

```
ARCCOS(X,PREC) ;
      ;
      New L,LIM,K,SIG,SIGS,VALUE
      ;
      If X<-1 Set $Ecode=",M28,"
      If X>1 Set $Ecode=",M28,"
      Set PREC=$Get(PREC,11)
      If $Translate(X,"-")=1 Quit 0
      ;
      Set SIG=$Select(X<0:-1,1:1),VALUE=1-(X*X)
      Set X=$%SQRT^MATH(VALUE,PREC)
      If $Translate(X,"-")=1 Do  Quit VALUE
      . Set VALUE=$%PI^MATH()/2*X
      . Quit
      ;
      If X>0.9 Do  Quit VALUE
      . Set SIGS=$Select(X<0:-1,1:1)
      . Set VALUE=1/(1/X/X-1)
      . Set X=$%SQRT^MATH(VALUE,PREC)
      . Set VALUE=$%ARCTAN^MATH(X,PREC)*SIGS
      . Quit
      Set (VALUE,L)=X
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
```

```
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)
      . Set L=L*X*X*(K-2)/(K-1)*(K-2)/K,VALUE=VALUE+L
      . Quit
      Quit $Select(SIG<0:$%PI^MATH()-VALUE,1:VALUE)
```

2.3 ARCCOSH

```
ARCCOSH(X,PREC) ;
      If X<1 Set $Ecode=",M28,"
      New SQ
      Set PREC=$Get(PREC,11)
      Set SQ=$%SQRT^MATH(X*X-1,PREC)
      Quit $%LOG^MATH(X+SQ,PREC)
```

2.4 ARCCOT

```
ARCCOT(X,PREC) ;
      Set PREC=$Get(PREC,11)
      Set X=1/X
      Quit $%ARCTAN^MATH(X,PREC)
```

2.5 ARCCOTH

```
ARCCOTH(X,PREC) ;
      New L1,L2
      Set PREC=$Get(PREC,11)
      Set L1=$%LOG^MATH(X+1,PREC)
      Set L2=$%LOG^MATH(X-1,PREC)
      Quit L1-L2/2
```

2.6 ARCCSC

```
ARCCSC(X,PREC) ;
      Set PREC=$Get(PREC,11)
      Set X=1/X
      Quit $%ARCSIN^MATH(X,PREC)
```

2.7 ARCSEC

```
ARCSEC(X,PREC) ;
      Set PREC=$Get(PREC,11)
      Set X=1/X
      Quit $%ARCCOS^MATH(X,PREC)
```

2.8 ARCSIN

Option 1, optimized for speed, not precision.

```
ARCSIN(X) ;
      ; This version of the function is
      ; optimized for speed, not for precision.
      ; The 'precision' parameter is not supported,
      ; and the precision is at best 2 in 10**-8.
      ;
      New A,N,R,SIGN,XX
      If X<-1 Set $Ecode=",M28,"
      If X>1 Set $Ecode=",M28,"
      Set SIGN=1 Set:X<0 X=-X,SIGN=-1
      Set A(0)=1.5707963050,A(1)=-0.2145988016,A(2)=0.0889789874
      Set A(3)=-0.0501743046,A(4)=0.0308918810,A(5)=-0.0170881256
      Set A(6)=0.0066700901,A(7)=-0.0012624911
```

```
      Set R=A(0),XX=1 For N=1:1:7 Set XX=XX*X,R=A(N)*XX+R
      Set R=$%SQRT^MATH(1-X,11)*R
      Set R=$%PI^MATH()/2-R
      Quit R*SIGN
```

Option 2, optimized for precision, not speed.

```
ARCSIN(X,PREC) ;
      New L,LIM,K,SIGS,VALUE
      Set PREC=$Get(PREC,11)
      If $Translate(X,"-")=1 Do  Quit VALUE
      . Set VALUE=$%PI^MATH()/2*X
      . Quit
      If X>0.99999 Do  Quit VALUE
      . Set SIGS=$Select(X<0:-1,1:1)
      . Set VALUE=1/(1/X/X-1)
      . Set X=$%SQRT^MATH(VALUE,PREC)
      . Set VALUE=$%ARCTAN^MATH(X,PREC)*SIGS
      . Quit
      Set (VALUE,L)=X
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)
      . Set L=L*X*X*(K-2)/(K-1)*(K-2)/K,VALUE=VALUE+L
      . Quit
      Quit VALUE
```

2.9 ARCSINH

```
ARCSINH(X,PREC) ;
      If X<1 Set $Ecode=",M28,"
      New SQ
      Set PREC=$Get(PREC,11)
      Set SQ=$%SQRT^MATH(X*X+1,PREC)
      Quit $%LOG^MATH(X+SQ,PREC)
```

2.10 ARCTAN

```
ARCTAN(X,PREC) ;
      New FOLD,HI,L,LIM,LO,K,SIGN,SIGS,SIGT,VALUE
      Set PREC=$Get(PREC,11)
      Set LO=0.0000000001,HI=9999999999
      Set SIGT=$Select(X<0:-1,1:1),X=$Translate(X,"-")
      Set X=$Select(X<LO:LO,X>HI:HI,1:X)
      Set FOLD=$Select(X'<1:0,1:1)
      Set X=$Select(FOLD:1/X,1:X)
      Set L=X,VALUE=$%PI^MATH()/2-(1/X),SIGN=1
      If X<1.3 Do  Quit VALUE
      . Set X=$Select(FOLD:1/X,1:X),VALUE=1/((1/X/X)+1)
      . Set X=$%SQRT^MATH(VALUE,PREC)
      . If $Translate(X,"-")=1 Do  Quit
      . . Set VALUE=$%PI^MATH()/2*X
      . . Quit
      . If X>0.9 Do  Quit
      . . Set SIGS=$Select(X<0:-1,1:1)
      . . Set VALUE=1/(1/X/X-1)
      . . Set X=$%SQRT^MATH(VALUE)
      . . Set VALUE=$$ARCTAN(X,10)
      . . Set VALUE=VALUE*SIGS
      . . Quit
      . Set (VALUE,L)=X
      . Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      . For K=3:2 Do  Quit:($Translate(L,"-")<LIM)
```

```
      . . Set L=L*X*X*(K-2)/(K-1)*(K-2)/K,VALUE=VALUE+L
      . . Quit
      . Set VALUE=$Select(SIGT<1:-VALUE,1:VALUE)
      . Quit
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(1/L,"-")<LIM)
      . Set L=L*X*X,VALUE=VALUE+(1/(K*L)*SIGN)
      . Set SIGN=SIGN*-1
      . Quit
      Set VALUE=$Select(FOLD:$%PI^MATH()/2-VALUE,1:VALUE)
      Set VALUE=$Select(SIGT<1:-VALUE,1:VALUE)
      Quit VALUE
```

2.11 ARCTANH

```
ARCTANH(X,PREC) ;
      If X<-1 Set $Ecode=",M28,"
      If X>1 Set $Ecode=",M28,"
      Set PREC=$Get(PREC,11)
      Quit $%LOG^MATH(1+X/(1-X),PREC)/2
```

2.12 CABS

```
CABS(Z) ;
      New ZRE,ZIM
      Set ZRE=+Z,ZIM=+$Piece(Z,"%",2)
      Quit $%SQRT^MATH(ZRE*ZRE+(ZIM*ZIM))
```

2.13 CADD

```
CADD(X,Y) ;
      New XRE,XIM,YRE,YIM
      Set XRE=+X,XIM=+$Piece(X,"%",2)
      Set YRE=+Y,YIM=+$Piece(Y,"%",2)
      Quit XRE+YRE_"%"_(XIM+YIM)
```

2.14 CCOS

```
CCOS(Z,PREC) ;
      New E1,E2,IA
      Set PREC=$Get(PREC,11)
      Set IA=$%CMUL^MATH(Z,"0%1")
      Set E1=$%CEXP^MATH(IA,PREC)
      Set IA=-IA_"%"_(-$Piece(IA,"%",2))
      Set E2=$%CEXP^MATH(IA,PREC)
      Set IA=$%CADD^MATH(E1,E2)
      Quit $%CMUL^MATH(IA,"0.5%0")
```

2.15 CDIV

```
CDIV(X,Y) ;
      New D,IM,RE,XIM,XRE,YIM,YRE
      Set XRE=+X,XIM=+$Piece(X,"%",2)
      Set YRE=+Y,YIM=+$Piece(Y,"%",2)
      Set D=YRE*YRE+(YIM*YIM)
      Set RE=XRE*YRE+(XIM*YIM)/D
      Set IM=XIM*YRE-(XRE*YIM)/D
      Quit RE_"%"_IM
```

2.16 CEXP

```
CEXP(Z,PREC) ;
      New R,ZIM,ZRE
      Set PREC=$Get(PREC,11)
      Set ZRE=+Z,ZIM=+$Piece(Z,"%",2)
      Set R=$%EXP^MATH(ZRE,PREC)
      Quit R*$%COS^MATH(ZIM,PREC)_"%"_(R*$%SIN^MATH(ZIM,PREC))
```

2.17 CLOG

```
CLOG(Z,PREC) ;
      New ABS,ARG,ZIM,ZRE
      Set PREC=$Get(PREC,11)
      Set ABS=$%CABS^MATH(Z)
      Set ZRE=+Z,ZIM=+$Piece(Z,"%",2)
      Set ARG=$%ARCTAN^MATH(ZIM/ZRE,PREC)
      Quit $%LOG^MATH(ABS,PREC)_"%"_ARG
```

2.18 CMUL

```
CMUL(X,Y) ;
      New XIM,XRE,YIM,YRE
      Set XRE=+X,XIM=+$Piece(X,"%",2)
      Set YRE=+Y,YIM=+$Piece(Y,"%",2)
      Quit XRE*YRE-(XIM*YIM)_"%"_(XRE*YIM+(XIM*YRE))
```

2.19 COMPLEX

```
COMPLEX(X) Quit +X_"%0"
```

2.20 CONJUG

```
CONJUG(Z) ;
      New ZIM,ZRE
      Set ZRE=+Z,ZIM=+$Piece(Z,"%",2)
      Quit ZRE_"%"_(-ZIM)
```

2.21 COS

Option 1, optimized for speed, not precision

```
COS(X) ;
      ; This version of the function is
      ; optimized for speed, not for precision.
      ; The 'precision' parameter is not supported,
      ; and the precision is at best 1 in 10**-9.
      ; Note that this function does not accept its
      ; parameter in degrees, minutes and seconds.
      ;
      New A,N,PI,R,SIGN,XX
      ;
      ; This approximation only works for 0 <= x <= pi/2
      ; so reduce angle to correct quadrant.
      ;
      Set PI=$%PI^MATH(),X=X#(PI*2),SIGN=1
      Set:X>PI X=2*PI-X
      Set:X*2>PI X=PI-X,SIGN=-1
      ;
      Set XX=X*X,A(1)=-0.4999999963,A(2)=0.0416666418
      Set A(3)=-0.0013888397,A(4)=0.0000247609,A(5)=-0.0000002605
```

```
      Set (X,R)=1 For N=1:1:5 Set X=X*XX,R=A(N)*X+R
      Quit R*SIGN
```

Option 2, optimized for precision, not speed.

```
COS(X,PREC) ;
      New L,LIM,K,SIGN,VALUE
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=1,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=2:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L*X*X/(K-1*K),VALUE=VALUE+(SIGN*L)
      . Quit
      Quit VALUE
```

2.22 COSH

```
COSH(X,PREC) ;
      New E,F,I,P,R,T,XX
      Set PREC=$Get(PREC,11)+1
      Set @("E=1E-"_PREC)
      Set XX=X*X,F=1,(P,R,T)=1,I=1
      For  Set T=T*XX,F=I+1*I*F,R=T/F+R,P=P-R/R,I=I+2 If -E<P,P<E Quit
      Quit R
```

2.23 COT

```
COT(X,PREC) ;
      New C,L,LIM,K,SIGN,VALUE
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      Set (VALUE,L)=1,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=2:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L*X*X/(K-1*K),VALUE=VALUE+(SIGN*L)
      . Quit
      Set C=VALUE
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=X,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L/(K-1)*X/K*X,VALUE=VALUE+(SIGN*L)
      . Quit
      If 'VALUE Quit "INFINITE"
      Quit VALUE=C/VALUE
```

2.24 COTH

```
COTH(X,PREC) ;
      New SINH
      If 'X Quit "INFINITE"
      ;
```

```
      Set PREC=$Get(PREC,11)
      Set SINH=$%SINH^MATH(X,PREC)
      If 'SINH Quit "INFINITE"
      Quit $%COSH^MATH(X,PREC)/SINH
```

2.25 CPOWER

```
CPOWER(Z,N,PREC) ;
      New AR,NIM,NRE,PHI,PI,R,RHO,TH,ZIM,ZRE
      ;
      Set PREC=$Get(PREC,11)
      Set ZRE=+Z,ZIM=+$Piece(Z,"%",2)
      Set NRE=+N,NIM=+$Piece(N,"%",2)
      If 'ZRE,'ZIM,'NRE,'NIM Set $Ecode=",M28,"
      ;
      If 'ZRE,'ZIM Quit "0%0"
      ;
      Set PI=$%PI^MATH()
      Set R=$%SQRT^MATH(ZRE*ZRE+(ZIM*ZIM),PREC)
      ;
      If ZRE Set TH=$%ARCTAN^MATH(ZIM/ZRE,PREC)
      Else  Set TH=$SELECT(ZIM>0:PI/2,1:-PI/2)
      ;
      Set RHO=$%LOG^MATH(R,PREC)
      Set AR=$%EXP^MATH(RHO*NRE-(TH*NIM),PREC)
      Set PHI=RHO*NIM+(NRE*TH)
      Quit AR*$%COS^MATH(PHI,PREC)_"%"_(AR*$%SIN^MATH(PHI,PREC))
```

2.26 CSC

```
CSC(X,PREC) ;
      New L,LIM,K,SIGN,VALUE
      ;
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      ;
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=X,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L/(K-1)*X/K*X,VALUE=VALUE+(SIGN*L)
      . Quit
      If 'VALUE Quit "INFINITE"
      Quit 1/VALUE
```

2.27 CSCH

```
CSCH(X,PREC) Quit 1/$%SINH^MATH(X,$Get(PREC,11))
```

2.28 CSIN

```
CSIN(Z,PREC) ;
      New IA,E1,E2
      ;
      Set PREC=$Get(PREC,11)
      ;
      Set IA=$%CMUL^MATH(Z,"0%1")
      Set E1=$%CEXP^MATH(IA,PREC)
```

```
      Set IA=-IA_"%"_(-$Piece(IA,"%",2))
      Set E2=$%CEXP^MATH(IA,PREC)
      Set IA=$%CSUB^MATH(E1,E2)
      Set IA=$%CMUL^MATH(IA,"0.5%0")
      Quit $%CMUL^MATH("0%-1",IA)
```

2.29 CSUB

```
CSUB(X,Y) ;
      New XIM,XRE,YIM,YRE
      Set XRE=+X,XIM=+$Piece(X,"%",2)
      Set YRE=+Y,YIM=+$Piece(Y,"%",2)
      Quit XRE-YRE_"%"_(XIM-YIM)
```

2.30 DECDMS

```
DECDMS(X,PREC) New T
      Set PREC=$Get(PREC,5)
      Set X=X#360*3600
      Set T=PREC-$Length(X\1)
      Set X=+$Justify(X,0,$Select(T'<0:T,1:0))
      Quit X\3600_":"_(X\60#60)_":"_(X#60)
```

2.31 DEGRAD

```
DEGRAD(X) Quit X*3.14159265358979/180
```

2.32 DMSDEC

```
DMSDEC(X) ;
      Quit $Piece(X,":")+($Piece(X,":",2)/60)+($Piece(X,":",3)/3600)
```

2.33 E

```
E() Quit 2.71828182845905
```

2.34 EXP

```
EXP(X,PREC) ;
      New L,LIM,K,VALUE
      Set PREC=$Get(PREC,11)
      Set L=X,VALUE=X+1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=2:1 Set L=L*X/K,VALUE=VALUE+L Quit:($Translate(L,"-")<LIM)
      Quit VALUE
```

2.35 LOG

```
LOG(X,PREC) ;
      New L,LIM,M,N,K,VALUE
      If X'>0 Set $Ecode=",M28,"
      Set PREC=$Get(PREC,11)
      Set M=1
      ;
      For N=0:1 Quit:(X/M)<10  Set M=M*10
      ;
      If X<1 For N=0:-1 Quit:(X/M)>0.1  Set M=M*0.1
      Set X=X/M
      Set X=(X-1)/(X+1),(VALUE,L)=X
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Set L=L*X*X,M=L/K,VALUE=M+VALUE Set:M<0 M=-M Quit:M<LIM
```

```
      Set VALUE=VALUE*2+(N*2.30258509298749)
      Quit VALUE
```

2.36 LOG10

```
LOG10(X,PREC) ;
      New L,LIM,M,N,K,VALUE
      If X'>0 Set $Ecode=",M28,"
      Set PREC=$Get(PREC,11)
      Set M=1
      For N=0:1 Quit:(X/M)<10  Set M=M*10
      If X<1 For N=0:-1 Quit:(X/M)>0.1  Set M=M*0.1
      Set X=X/M
      Set X=(X-1)/(X+1),(VALUE,L)=X
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Set L=L*X*X,M=L/K,VALUE=M+VALUE Set:M<0 M=-M Quit:M<LIM
      Set VALUE=VALUE*2+(N*2.30258509298749)
      Quit VALUE/2.30258509298749
```

2.37 MTXADD

```
MTXADD(A,B,R,ROWS,COLS) ;
      ; Add A[ROWS,COLS] to B[ROWS,COLS],
      ; result goes to R[ROWS,COLS]
      If $Data(A)<10 Quit 0
      If $Data(B)<10 Quit 0
      If $Get(ROWS)<1 Quit 0
      If $Get(COLS)<1 Quit 0
      ;
      New ROW,COL,ANY
      For ROW=1:1:ROWS For COL=1:1:COLS Do
      . KValue R(ROW,COL) Set ANY=0
      . Set:$Data(A(ROW,COL))#2 ANY=1
      . Set:$Data(B(ROW,COL))#2 ANY=1
      . Set:ANY R(ROW,COL)=$Get(A(ROW,COL))+$Get(B(ROW,COL))
      . Quit
      Quit 1
```

2.38 MTXCOF

```
MTXCOF(A,I,K,N) ;
      ; Compute cofactor for element [i,k]
      ; in matrix A[N,N]
      New T,R,C,RR,CC
      Set CC=0 For C=1:1:N Do:C'=K
      . Set CC=CC+1,RR=0
      . For R=1:1:N Set:R'=I RR=RR+1,T(RR,CC)=$Get(A(R,C))
      . Quit
      Quit $%MTXDET^MATH(.T,N-1)
```

2.39 MTXCOPY

```
MTXCOPY(A,R,ROWS,COLS) ;
      ; Copy A[ROWS,COLS] to R[ROWS,COLS]
      If $Data(A)<10 Quit 0
      If $Get(ROWS)<1 Quit 0
      If $Get(COLS)<1 Quit 0
      ;
      New ROW,COL
      For ROW=1:1:ROWS For COL=1:1:COLS Do
      . KValue R(ROW,COL)
      . Set:$Data(A(ROW,COL))#2 R(ROW,COL)=A(ROW,COL)
```

```
      . Quit
      Quit 1
```

2.40 MTXDET

```
MTXDET(A,N) ;
      ; Compute determinant of matrix A[N,N]
      If $Data(A)<10 Quit ""
      If $Get(N)<1 Quit ""
      ;
      ; First the simple cases
      ;
      If N=1 Quit $Get(A(1,1))
      If N=2 Quit $Get(A(1,1))*$Get(A(2,2))-($Get(A(1,2))*$Get(A(2,1)))
      ;
      New DET,I,SIGN
      ;
      ; Det A = sum (k=1:n) element (i,k) * cofactor [i,k]
      ;
      Set DET=0,SIGN=1
      For I=1:1:N Do
      . Set DET=$Get(A(1,I))*$%MTXCOF^MATH(.A,1,I,N)*SIGN+DET
      . Set SIGN=-SIGN
      . Quit
      Quit DET
```

2.41 MTXEQU

```
MTXEQU(A,B,R,N,M) ;
      ; Solve matrix equation A [M,M] * R [M,N] = B [M,N]
      If $Get(M)<1 Quit ""
      If $Get(N)<1 Quit ""
      If '$%MTXDET^MATH(.A,M) Quit 0
      ;
      New I,I1,J,J1,J2,K,L,T,T1,T2,TEMP,X
      ;
      Set X=$%MTXCOPY^MATH(.A,.T,N,N)
      Set X=$%MTXCOPY^MATH(.B,.R,N,M)
      ;
      ; Reduction of matrix A
      ; Steps of reduction are counted by index K
      ;
      For K=1:1:N-1 Do
      . ;
      . ; Search for largest coefficient of T
      . ; (denoted by TEMP)
      . ; in first column of reduced system
      . ;
      . Set TEMP=0,J2=K
      . For J1=K:1:N Do
      . . Quit:$TRanslate($Get(T(J1,K)),"-")>$TRanslate(TEMP,"-")
      . . Set TEMP=T(J1,K),J2=J1
      . . Quit
      . ;
      . ; Exchange row number K with row number J2,
      . ; if necessary
      . ;
      . Do:J2'=K
      . . ;
      . . For J=K:1:N Do
      . . . Set T1=$Get(T(K,J)),T2=$Get(T(J2,J))
      . . . Kill T(K,J),T(J2,J)
      . . . If T1'="" Set T(J2,J)=T1
```

```
      . . . If T2'="" Set T(K,J)=T2
      . . . Quit
      . . For J=1:1:M Do
      . . . Set T1=$Get(R(K,J)),T2=$Get(R(J2,J))
      . . . Kill R(K,J),R(J2,J)
      . . . If T1'="" Set R(J2,J)=T1
      . . . If T2'="" Set R(K,J)=T2
      . . . Quit
      . . Quit
      . ;
      . ; Actual reduction
      . ;
      . For I=K+1:1:N Do
      . . For J=K+1:1:N Do
      . . . Quit:'$Get(T(K,K))
      . . . Set T(I,J)=-$Get(T(K,J))*$Get(T(I,K))/T(K,K)+$Get(T(I,J))
      . . . Quit
      . . For J=1:1:M Do
      . . . Quit:'$Get(T(K,K))
      . . . Set R(I,J)=-$Get(R(K,J))*$Get(T(I,K))/T(K,K)+$Get(R(I,J))
      . . . Quit
      . . Quit
      . Quit
      ;
      ; Backsubstitution
      ;
      For J=1:1:M Do
      . If $Get(T(N,N)) Set R(N,J)=$Get(R(N,J))/T(N,N)
      . If N-1>0 For I1=1:1:N-1 Do
      . . Set I=N-I1
      . . For L=I+1:1:N Do
      . . . Set R(I,J)=-$Get(T(I,L))*$Get(R(L,J))+$Get(R(I,J))
      . . . Quit
      . . If $Get(T(I,I)) Set R(I,J)=$Get(R(I,J))/$Get(T(I,I))
      . . Quit
      . Quit
      Quit $Select(M=N:$%MTXDET^MATH(.R,M),1:1)
```

2.42 MTXINV

```
MTXINV(A,R,N) ;
      ; Invert A[N,N], result goes to R[N,N]
      If $Data(A)<10 Quit 0
      If $Get(N)<1 Quit 0
      ;
      New T,X
      Set X=$%MTXUNIT^MATH(.T,N)
      Quit $%MTXEQU^MATH(.A,.T,.R,N,N)
```

2.43 MTXMUL

```
MTXMUL(A,B,R,M,L,N) ;
      ; Multiply A[M,L] by B[L,N], result goes to R[M,N]
      If $Data(A)<10 Quit 0
      If $Data(B)<10 Quit 0
      If $Get(L)<1 Quit 0
      If $Get(M)<1 Quit 0
      If $Get(N)<1 Quit 0
      ;
      New I,J,K,SUM,ANY
      For I=1:1:M For J=1:1:N Do
      . Set (SUM,ANY)=0
      . KValue R(I,J)
```

```
      . For K=1:1:L Do
      . . Set:$Data(A(I,K))#2 ANY=1
      . . Set:$Data(B(K,J))#2 ANY=1
      . . Set SUM=$Get(A(I,K))*$Get(B(K,J))+SUM
      . . Quit
      . Set:ANY R(I,J)=SUM
      . Quit
      Quit 1
```

2.44 MTXSCA

```
MTXSCA(A,R,ROWS,COLS,S) ;
      ; Multiply A[ROWS,COLS] with the scalar S,
      ; result goes to R[ROWS,COLS]
      If $Data(A)<10 Quit 0
      If $Get(ROWS)<1 Quit 0
      If $Get(COLS)<1 Quit 0
      If '($Data(S)#2) Quit 0
      ;
      New ROW,COL
      For ROW=1:1:ROWS For COL=1:1:COLS Do
      . KValue R(ROW,COL)
      . Set:$Data(A(ROW,COL))#2 R(ROW,COL)=A(ROW,COL)*S
      . Quit
      Quit 1
```

2.45 MTXSUB

```
MTXSUB(A,B,R,ROWS,COLS) ;
      ; Subtract B[ROWS,COLS] from A[ROWS,COLS],
      ; result goes to R[ROWS,COLS]
      If $Data(A)<10 Quit 0
      If $Data(B)<10 Quit 0
      If $Get(ROWS)<1 Quit 0
      If $Get(COLS)<1 Quit 0
      ;
      New ROW,COL,ANY
      For ROW=1:1:ROWS For COL=1:1:COLS Do
      . KValue R(ROW,COL) Set ANY=0
      . Set:$Data(A(ROW,COL))#2 ANY=1
      . Set:$Data(B(ROW,COL))#2 ANY=1
      . Set:ANY R(ROW,COL)=$Get(A(ROW,COL))-$Get(B(ROW,COL))
      . Quit
      Quit 1
```

2.46 MTXTRP

```
MTXTRP(A,R,M,N) ;
      ; Transpose A[M,N], result goes to R[N,M]
      If $Data(A)<10 Quit 0
      If $Get(M)<1 Quit 0
      If $Get(N)<1 Quit 0
      ;
      New I,J,K,D1,V1,D2,V2
      For I=1:1:M+N-1 For J=1:1:I+1\2 Do
      . Set K=I-J+1
      . If K=J Do  Quit
      . . Set V1=$Get(A(J,J)),D1=$Data(A(J,J))#2
      . . If J'>N,J'>M KValue R(J,J) Set:D1 R(J,J)=V1
      . . Quit
      . ;
      . Set V1=$Get(A(K,J)),D1=$Data(A(K,J))#2
```

```
      . Set V2=$Get(A(J,K)),D2=$Data(A(J,K))#2
      . If K'>M,J'>N KValue R(K,J) Set:D2 R(K,J)=V2
      . If J'>M,K'>N KValue R(J,K) Set:D1 R(J,K)=V1
      . Quit
      Quit 1
```

2.47 MTXUNIT

```
MTXUNIT(R,N,SPARSE) ;
      ; Create a unit matrix R[N,N]
      If $Get(N)<1 Quit 0
      ;
      New ROW,COL
      For ROW=1:1:N For COL=1:1:N Do
      . KValue R(ROW,COL)
      . If $Get(SPARSE) Quit:ROW'=COL
      . Set R(ROW,COL)=$Select(ROW=COL:1,1:0)
      . Quit
      Quit 1
```

2.48 PI

```
PI() Quit 3.14159265358979
```

2.49 RADDEG

```
RADDEG(X) Quit X*180/3.14159265358979
```

2.50 SEC

```
SEC(X,PREC) ;
      New L,LIM,K,SIGN,VALUE
      ;
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=1,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=2:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L*X*X/(K-1*K),VALUE=VALUE+(SIGN*L)
      . Quit
      If 'VALUE Quit "INFINITE"
      Quit 1/VALUE
```

2.51 SECH

```
SECH(X,PREC) Quit 1/$%COSH^MATH(X,$Get(PREC,11))
```

2.52 SIGN

```
SIGN(X) Quit $Select(X<0:-1,X>0:1,1:0)
```

2.53 SIN

Option 1, optimized for speed, not precision.

```
SIN(X) ;
```

```
      ; This version of the function is
      ; optimized for speed, not for precision.
      ; The 'precision' parameter is not supported,
      ; and the precision is at best 1 in 10**-9.
      ; Note that this function does not accept its
      ; parameter in degrees, minutes and seconds.
      ;
      New A,N,PI,R,SIGN,XX
      ;
      ; This approximation only works for 0 <= x <= pi/2
      ; so reduce angle to correct quadrant.
      ;
      Set PI=$%PI^MATH(),X=X#(PI*2),SIGN=1
      Set:X>PI X=2*PI-X,SIGN=-1
      Set:X*2<PI X=PI-X
      Set XX=X*X,A(1)=-0.4999999963,A(2)=0.0416666418
      Set A(3)=-0.0013888397,A(4)=0.0000247609,A(5)=-0.0000002605
      Set (X,R)=1 For N=1:1:5 Set X=X*XX,R=A(N)*X+R
      Quit R*SIGN
```

Option 2, optimized for precision, not speed

```
SIN(X,PREC) ;
      New L,LIM,K,SIGN,VALUE
      ;
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=X,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L/(K-1)*X/K*X,VALUE=VALUE+(SIGN*L)
      . Quit
      Quit VALUE
```

2.54 SINH

```
SINH(X,PREC) ;
      New E,F,I,P,R,T,XX
      ;
      Set PREC=$Get(PREC,11)+1
      Set @("E=1E-"_PREC)
      Set XX=X*X,F=1,I=2,(P,R,T)=X
      For  Set T=T*XX,F=I+1*I*F,R=T/F+R,P=P-R/R,I=I+2 If -E<P,P<E Quit
      Quit R
```

2.55 SQRT

```
SQRT(X,PREC) ;
      If X<0 Set $Ecode=",M28,"
      If X=0 Quit 0
      ;
      Set PREC=$Get(PREC,11)
      If X<1 Quit 1/$%SQRT^MATH(1/X,PREC)
      ;
      New P,R,E
      Set PREC=$Get(PREC,11)+1
      Set @("E=1E-"_PREC)
```

```
      ;
      Set R=X
      For  Set P=R,R=X/R+R/2,P=P-R/R If -E<P,P<E Quit
      Quit R
```

2.56 TAN

```
TAN(X,PREC) ;
      New L,LIM,K,S,SIGN,VALUE
      ;
      ; The official description does not mention that
      ; the function may also be called with the first
      ; parameter in degrees, minutes and seconds.
      Set:X[":" X=$%DMSDEC^MATH(X)
      ;
      Set PREC=$Get(PREC,11)
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=X,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=3:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L/(K-1)*X/K*X,VALUE=VALUE+(SIGN*L)
      . Quit
      Set S=VALUE
      Set X=X#(2*$%PI^MATH())
      Set (VALUE,L)=1,SIGN=-1
      Set LIM=$Select((PREC+3)'>11:PREC+3,1:11),@("LIM=1E-"_LIM)
      For K=2:2 Do  Quit:($Translate(L,"-")<LIM)  Set SIGN=SIGN*-1
      . Set L=L*X*X/(K-1*K),VALUE=VALUE+(SIGN*L)
      . Quit
      If 'VALUE Quit "INFINITE"
      Quit S/VALUE
```

2.57 TANH

```
TANH(X,PREC) ;
      Set PREC=$Get(PREC,11)
      ;
      Quit $%SINH^MATH(X,PREC)/$%COSH^MATH(X,PREC)
```

3 STRING Library

3.1 CRC16

```
CRC16(string,seed) ;
      ;
      ; The code below was approved in document X11/1998-32
      ;
      ; Polynomial x**16 + x**15 + x**2 + x**0
      New I,J,R
      If '$Data(seed) Set R=0
      Else  If seed'<0,seed'>65535 Set R=seed\1
      Else  Set $ECode=",M28,"
      For I=1:1:$Length(string) Do
      . Set R=$$XOR($Ascii(string,I),R,8)
      . For J=0:1:7 Do
      . . If R#2 Set R=$$XOR(R\2,40961,16)
      . . Else  Set R=R\2
      . . Quit
      . Quit
```

```
      Quit R
XOR(a,b,w) New I,M,R
      Set R=b,M=1
      For I=1:1:w Do
      . Set:a\M#2 R=R+$Select(R\M#2:-M,1:M)
      . Set M=M+M
      . Quit
      Quit R
```

3.2 CRC32

```
CRC32(string,seed) ;
      ;
      ; The code below was approved in document X11/1998-32
      ;
      ; Polynomial X**32 + X**26 + X**23 + X**22 +
      ;          + X**16 + X**12 + X**11 + X**10 +
      ;          + X**8  + X**7  + X**5  + X**4 +
      ;          + X**2  + X     + 1
      New I,J,R
      If '$Data(seed) Set R=4294967295 ; = 2**32 - 1
      Else  If seed'<0,seed'>4294967295 Set R=4294967295-seed
      Else  Set $ECode=",M28,"
      For I=1:1:$Length(string) Do
      . Set R=$$XOR($Ascii(string,I),R,8)
      . For J=0:1:7 Do
      . . If R#2 Set R=$$XOR(R\2,3988292384,32)
      . . Else  Set R=R\2
      . . Quit
      . Quit
      Quit 4294967295-R ; 32-bit ones complement
XOR(a,b,w) New I,M,R
      Set R=b,M=1
      For I=1:1:w Do
      . Set:a\M#2 R=R+$Select(R\M#2:-M,1:M)
      . Set M=M+M
      . Quit
      Quit R
```

3.3 CRCCCITT

```
CRCCCITT(string,seed) ;
      ;
      ; The code below was approved in document X11/1998-32
      ;
      ; Polynomial x**16 + x**12 + x**5 + x**0
      New I,J,R
      If '$Data(seed) Set R=65535
      Else  If seed'<0,seed'>65535 Set R=seed\1
      Else  Set $ECode=",M28,"
      For I=1:1:$Length(string) Do
      . Set R=$$XOR($Ascii(string,I)*256,R,16)
      . For J=0:1:7 Do
      . . Set R=R+R
      . . Quit:R<65536
      . . Set R=$$XOR(4129,R-65536,13)
      . . Quit
      . Quit
      Quit R
XOR(a,b,w) New I,M,R
      Set R=b,M=1
      For I=1:1:w Do
      . Set:a\M#2 R=R+$Select(R\M#2:-M,1:M)
```

```
      . Set M=M+M
      . Quit
      Quit R
```

3.4 FORMAT

```
FORMAT(V,S) ;
      ;
      ; The code below was approved in document X11/SC13/TG2/1999-1
      ;
      New lo,mask,out,p,pos,spec,up,v1,v2,val,x
      ;
      Set lo="abcdefghijklmnopqrstuvwxyz"
      Set up="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      ;
      ; Array spec() contains the formatting directives
      ;
      ; First set defaults
      ;
      Set spec("CS")="$" ; Currency symbol
      Set spec("DC")="." ; Decimal separator
      Set spec("EC")="*" ; Error character
      Set spec("SL")="," ; Separator characters > 1
      Set spec("FS")=" " ; Fill string
      ;
      ; Other specifiers may be
      ;  FM = Format Mask
      ;  FO = Fill On/Off
      ;  SR = Separator characters < 1
      ;
      ; Then Inherit properties from System,
      ; overwriting the defaults
      ;
      Set x="" For  Set x=$Order(^$System($System,"FORMAT",x)) Quit:x=""  Do
      . Set spec(x)=^$System($System,"FORMAT",x)
      . Quit
      ;
      ; Then Inherit properties from current process
      ; overwriting the system and the defaults
      ;
      Set x="" For  Set x=$Order(^$Job($Job,"FORMAT",x)) Quit:x=""  Do
      . Set spec(x)=^$Job($Job,"FORMAT",x)
      . Quit
      ;
      ; Then look at actual parameters
      ; overwriting anything else
      ;
      Set S=$Get(S) For  Quit:S=""  Do
      . New e,i,str,v
      . Set x=$Piece(S,"=",1)
      . Set i=$Length(x)+2,str=0,v=""
      . Set:x="" i=1
      . For i=i:1:$Length(S)+1 Do  Quit:'i
      . . Set e=$Extract(S_":",i)
      . . If 'str,e=":" Set S=$Extract(S,i+1,$Length(S)),i=0 Quit
      . . Set v=v_e Quit:e'=""""
      . . Set str=1-str
      . . Quit
      . If i>$Length(S) Set S=""
      . If x'="",v'="" Set @("spec($Translate(x,lo,up))="_v) Quit
      . Set $ECode=",M28,"
      . Quit
      ;
```

```
	; Make certain that DC and EC are non-empty
	; and not longer than 1 character
	;
	Set spec("DC")=$Extract(spec("DC")_".",1)
	Set spec("EC")=$Extract(spec("EC")_"*",1)
	;
	Set val=$Get(V),(mask,out)=$Get(spec("FM"))
	If mask="" Quit val
	;
	; Currency string
	;
	Set x=spec("CS")
	Set pos=0 For  Set pos=$Find(mask,"c",pos) Quit:pos<1  Do
	. Set $Extract(out,pos-1)=$Extract(x,1)
	. Set x=$Extract(x,2,$Length(x))_$Extract(x,1)
	. Quit
	;
	; Sign
	;
	Set x=$Select(val>0:"+",val<0:"-",1:" ")
	Set pos=0 For  Set pos=$Find(mask,"+",pos) Quit:pos<1  Do
	. Set $Extract(out,pos-1)=x
	. Quit
	Set pos=0 For  Set pos=$Find(mask,"-",pos) Quit:pos<1  Do
	. Set $Extract(out,pos-1)=$Select(x="-":x,1:" ")
	. Quit
	If x'="-" Set out=$Translate(out,"()","  ")
	;
	; Decimal separator
	;
	Set pos=$Find(mask,"d")
	Do:pos'<1
	. Set $Extract(out,pos-1)=spec("DC")
	. For  Set pos=$Find(mask,"d",pos) Quit:pos<1  Do
	. . Set $Extract(out,pos-1)=spec("EC")
	. . Quit
	. Quit
	;
	; Right (default, format letter "n") or
	; left (format letter "l") adjustment?
	;
	If mask["l",mask["n" Set $ECode=",M28,"
	;
	; Left and Right Separators
	;
	Set v1=$Piece(val,".",1),v2=$Piece(val,".",2)
	Set v1=$Translate(v1,"-")
	If mask'["l" Do
	. Set x="" For p=1:1:$Length(v1) Set x=$Extract(v1,p)_x
	. Set v1=x
	. Quit
	;
	Set pos=$Find(mask,"d") Set:pos<1 pos=$Length(mask)+2
	;
	; Integer part and Left separators
	;
	Set x=spec("SL")
	Set p(1)=pos-2,p(2)=-1,p(3)=1
	Set:mask["l" p(1)=1,p(2)=1,p(3)=pos-2
	For p=p(1):p(2):p(3) Do
	. If "fln"[$Extract(mask,p) Do
	. . Set $Extract(out,p)=$Extract(v1,1)
	. . Set v1=$Extract(v1,2,$Length(v1))_spec("FS")
```

```
      . . If $Translate(v1,spec("FS"))="" Set x=spec("FS")
      . . Quit
      . If $Extract(mask,p)="s" Do
      . . Set $Extract(out,p)=$Extract(x,1)
      . . Set x=$Extract(x,2,$Length(x))_$Extract(x,1)
      . Quit
      ;
      ; Fractional part and Right separators
      ;
      Set x=$Get(spec("SR"),spec("SL"))
      Set:v2="" v2=0
      For p=pos:1:$Length(mask) Do
      . If "fn"[$Extract(mask,p) Do
      . . Set $Extract(out,p)=$Extract(v2,1)
      . . Set v2=$Extract(v2,2,$Length(v2))_"0"
      . . Quit
      . If $Extract(mask,p)="s" Do
      . . Set $Extract(out,p)=$Extract(x,1)
      . . Set x=$Extract(x,2,$Length(x))_$Extract(x,1)
      . . Quit
      . Quit
      ;
      ; Fill String
      ;
      Set x=$Get(spec("FS"))
      For p=1:1:$l(mask) Do
      . Quit:"nf"'[$Extract(mask,p)
      . Quit:$Extract(out,p)'=" "
      . Set $Extract(out,p)=$Extract(x,1)
      . Set x=$Extract(x,2,$Length(x))_$Extract(x,1)
      . Quit
      ;
      ; Justification
      ;
      For x="+ | +","- | -","( | ("," )|) " Do
      . New find,repl
      . Set find=$Piece(x,"|",1),repl=$Piece(x,"|",2)
      . For  Quit:out'[find  Do
      . . Set out=$Piece(out,find,1)_repl_$Piece(out,find,2,$l(out)+2)
      . . Quit
      . Quit
      ;
      Quit out
```

3.5 PRODUCE

```
PRODUCE(IN,SPEC,MAX) ;
      New VALUE,AGAIN,P1,P2,I,COUNT
      Set VALUE=IN,COUNT=0
      For  Do  Quit:'AGAIN
      . Set AGAIN=0
      . Set I=""
      . For  Set I=$Order(SPEC(I)) Quit:I=""  Do  Quit:COUNT<0
      . . Quit:$Get(SPEC(I,1))=""
      . . Quit:'($Data(SPEC(I,2))#2)
      . . For  Quit:VALUE'[SPEC(I,1)  Do  Quit:COUNT<0
      . . . Set P1=$PIECE(VALUE,SPEC(I,1),1)
      . . . Set P2=$PIECE(VALUE,SPEC(I,1),2,$Length(VALUE))
      . . . Set VALUE=P1_SPEC(I,2)_P2,AGAIN=1
      . . . Set COUNT=COUNT+1
      . . . If $Data(MAX),COUNT>MAX Set COUNT=-1,AGAIN=0
      . . . Quit
      . . Quit
```

```
      . Quit
      Quit VALUE
```

3.6 REPLACE

```
REPLACE(IN,SPEC) ;
      New L,MASK,K,I,LT,F,VALUE
      Set L=$Length(IN),MASK=$JUSTIfY("",L)
      Set I="" For  Set I=$Order(SPEC(I)) Quit:I=""  Do
      . Quit:'($Data(SPEC(I,1))#2)
      . Quit:SPEC(I,1)=""
      . Quit:'($Data(SPEC(I,2))#2)
      . Set LT=$Length(SPEC(I,1))
      . Set F=0 For  Set F=$Find(IN,SPEC(I,1),F) Quit:F<1  Do
      . . Quit:$Extract(MASK,F-LT,F-1)["X"
      . . Set VALUE(F-LT)=SPEC(I,2)
      . . Set $Extract(MASK,F-LT,F-1)=$TRanslate($Justify("",LT)," ","X")
      . . Quit
      . Quit
      Set VALUE="" For K=1:1:L Do
      . If $Extract(MASK,K)=" " Set VALUE=VALUE_$Extract(IN,K) Quit
      . Set:$Data(VALUE(K)) VALUE=VALUE_VALUE(K)
      . Quit
      Quit VALUE
```

Index