1. Identification of the Proposed Change

1.1 Title

Error Handling Corrections

1.2 MDC Proposer and Sponsor

Proposer
David J. Marcus
Micronetics Design Corp.
1375 Piccard Drive
Rockville, MD 20850
301-258-2605
davidm@micronetics.com

Sponsor
Wally Fort
VA ISC-SF
301 Howard St., Suite 600
San Francisco, CA. 94105
(415) 744-7520
fort.w@forum.va.gov

1.3 Motion

That this document X11/SC15/98-5 be adopted as a MDC Type A document superseding X11/SC15/TG17/98-2.

1.4 History

June 1998	X11/SC15/98-5	<this document>
March 1998	X11/SC15/TG17/98-2	Passed as a SC15 type A document (18:0:2)
Sept. 1997	X11/SC15/TG17/97-5	Reaffirmed as SC15 type B after being amended in task group. Passed (21:0:3)
Mar. 1997	X11/SC15/TG17/97-1	SC15 Type B, Passed (23:0:3)
Sept. 1996	X11/SC15/TG17/96-5	New document editor Revised definition of $STACK(-1), added an annex.
Oct. 1995	X11/SC15/95-38	SC Type B Passed (17-2-6)
May 1995	X11/SC15/	This document proposed for Type C approval.

1.5 Dependencies

This proposal is dependent upon: None Known

Proposals that depend on this proposal: None Known

2. Justification of the Proposed Change

2.1 Needs

Several errors, ambiguities and inconsistencies have been identified (see X11/94-49) in the existing Error Handling portion of the X11.1 document. This proposal attempts to rectify them. The relevant contents of X11/94-49 are reproduced below:

1. Section 7.1.5.19, single operand version of $STACK(), says that "a) if intexpr is -1, returns the largest value of intexpr for which the $STACK functions will return a non-empty value." This is ambiguous: which form of the $STACK function, the single or double operand version? If it is the two-operand version, then which value (ECODE, MCODE, or PLACE)?

2. Section 7.1.5.19, single operand version of $STACK(), says that "a) if intexpr is -1, returns the largest value of intexpr for which the $STACK functions

will return a non-empty value." It also says "d) if intexpr is greater than $STACK(-1), returns an empty string". This appears to be a major case of circular reasoning.

3. In section 7.1.5.19 (pg. 37), the two argument form of $STACK, states: "This form returns information about the action that created this level ...". This section does not define what 'this level' is. I think it meant to say "that created the level identified by intexpr". As it stands, it is ambiguous.

4. In section 7.1.5.19, two argument form of $STACK, for ECODE, states: "the list of any ecodes added at this level". This does not define what "this level' is. As it stands, it is ambiguous.

5. In section 7.1.5.19: Bottom of page: Offers three cases for identifying where "PLACE" is. These cases do not cover the case of intexpr > $STACK and the case of intexpr < 0. This leaves two interpretations possible: either the value is then up to the implementation or it is an error.

6. The portability standard is silent on the maximum length of $ETRAP. Is it covered by the general limit on length of values?

7. Section 8.2.18, SET command, states that "if the value of $ECODE becomes non-empty, an error trap will be invoked." Does SET $EC=$EC, with $EC already non-empty, trigger an error trap? The ambiguity is the use of the words "becomes non-empty". This could be interpreted as either it is non-empty at the end of the operation regardless of its original value, or it can be interpreted as a transition to non-empty. As it stands, this is ambiguous.

8. In section 8.2.18, SET command, states that resetting $ECODE to the empty string makes $STACK($STACK,"ECODE") return the empty string and "as do all forms of the function $STACK($STACK+*n*) for all values of *n* greater than 0." This specifically does not say anything about the two operand version $STACK() for all parameter values other than "STACK,"ECODE". When does the other $STACK values get cleared?

9. In section 7.1.4.10, page 25: The description for $EC[ODE] says "This information is loaded by the implementation after detecting an erroneous condition, or by the application via the SET command." This is somewhat different then what the original proposal said in its section 3.1, that "1) When MUMPS encounters an error it appends the code for that error to $ECODE". The standard's use of the words 'loaded by the implementation ... or via SET ..." implies that the new error code *replaces* the existing value rather than appending to it. Which is it, does it append or replace? (the question is actually rhetorical since the Standard should rule supreme).

10. If we take the interpretation that the new error code(s) are appended to the existing value of $ECODE when an error is detected by M, then the portability standard is silent on the maximum length to which $ECODE could grow. In addition the portability standard needs to specify what happens

when the maximum length is exceeded. For example, is there another error generated? Does it replace the existing $ECODE? Is the existing $ECODE truncated? Etc.

11. If we take the interpretation that new error code(s) are appended: then what happens when the same error code is generated twice at the same level (as could happen for example with a division by zero, error trap trying to correct at the same level and getting another division by zero). Does $STACK(level,"ECODE") contain the error twice? This question is also motivated by the description of "ECODE" under section 7.1.5.19 for $STACK(intexpr,stackcodeexpr). It says (near the bottom of page 37): "the list of any ecodes added at this level." This explanation implies (to me, at least) that if the level already contained a specific ecode, it does not get added again. Worst case, this is ambiguous.

12. Section 6.3.2, last paragraph, states (pg. 11): "When $ST($ST,"ECODE") returns a non-empty string and the value of $ECODE changes to a non-empty string, ...". The issue is with the word "changes". If $EC is non-empty, does "SET $EC=$EC" force setting $STACK($STACK+1,...) as required by section 6.3.2? Or can it be interpreted that $ECODE did not change (it certainly has the same value before and after the SET)?

13. Section 6.3.2, last paragraph, discusses what happens when an error occurs in the error handling M code. It states (pg. 11): "a) It associates the $STACK information about the failure as if it were associated with the frame identified by $STACK+1". The issue here is: What are the values of the "MCODE" and "PLACE" nodes of $STACK($STACK+1,...) ?

14. Section 6.3.2 describes how $ETRAP is invoked. It states (bottom page 10): "An Error Processing transfer of control consists of an implicit GOTO [...] to the following two lines of code [...] . These lines are implicitly incorporated into the current execution environment immediately preceding the next command in the normal execution sequence."

    a) There are three separate issues with the 'placement' of the $ETRAP code lines. The first is the implication that the $ETRAP code is to be inserted after the current command of the line. That is, if the error occurs in the 3rd command of a 5 command line, the $ETRAP code appears as the 4th command on the line (which is immediately "preceding the next command in the normal execution sequence"). However, if the two lines are inserted given the format specified in the section, the result would be syntactically erroneous. [Unless, of course, the insertion splits the current line into two lines and inserts the added lines in-between the two fragments]. [The correct wording should have been to implicitly insert a GOTO with a dummy label to the end of the routine where the two lines are appended. Of course, implicit insertion should ignore line length portability limits!!]

    b) The second problem with the insertion has to do with line numbering. Assuming that the intent is to simply insert the two lines after the current line, what happens to line numbering? If the error occurred at line LABEL+1 (meaning the $ETRAP code is inserted as LABEL+2 and LABEL+3),

what happens if the $ETRAP code contains a GOTO LABEL+2? Does it go to the original LABEL+2 line or the current LABEL+2 after insertion?

c) The third (and most significant) issue has to do with Structured DO levels. The inserted $ETRAP code lines are at linelevel of 0. That means, the code cannot go back into the structured DO. They should be inserted with the current linelevel.

2.2 Existing Practice in Area of the Proposed Change

Some vendors have tried to implement a self consistent version of Error Handling. Others, not realizing the errors were there, implemented the "intentions" of the original proposal defined in the non-technical portion of the original proposal.

3. Description of the Proposed Change

3.1 General Description of the Proposed Change

Correct the identified problems.

3.2 Annotated Examples of Use

N/A.

3.3 Formalization

Each of the problems identified in section 2.1 (Needs) above will be referred to as problem N2.1.n where the 'n' is the subs-section number in the Needs section 2.1 above.

N2.1.1

In section "7.1.5.19a)" Replace the entire paragraph with:

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.

N2.1.2

In section "7.1.5.19c)" and "7.1.5.19d)" Change $STACK(-1) to $STACK

N2.1.3

In section 7.1.5.19, immediately after the BNF for the two argument form of $ST[ACK], replace the sentence beginning with 'This form returns information...' with:

This form returns information about the action that created the level of the PROCESS-STACK identified by intexpr as follows:

N2.1.4

In section 7.1.5.19, replace the sentence defining the 'Returned String' for ECODE with:

The list of any ecodes added at the level of the PROCESS-STACK identified by intexpr.

N2.1.5

In section 7.1.5.19, add the following cases for values of intexpr under the two-argument form of $STACK.

a) Values of intexpr < 0 are reserved.

b) Values of intexpr > $STACK return the empty string.

N2.1.6

Upon closer review of the M Portability Requirements, section 2.8 Character Strings, no changes are required. This section does not *explicitly* indicate that it does not apply to $ETRAP and hence, in the absence of any other indications elsewhere in the Standard, it must be assumed that this section does apply.

N2.1.7 and N2.1.8

In section 8.2.18, SET command description of updating $ECODE, replace section with:

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 format required in section 7.1.4.10 for $ECODE, the SET of $ECODE to the value of the expr is not performed. An "M101" error is generated instead.

2) If the value of expr does conform to format required in section 7.1.4.10 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 which is updating $ECODE.

d) The value of $STACK($STACK,"MCODE") is replaced to reflect the SET command which is updating $ECODE.

e) An error trap is invoked.

N2.1.9 and N2.1.10

In section 6.3.2 Error Processing, add the following paragraph immediately in front of "An Error Processing transfer of control is performed when:"

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 implementations maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,"ECODE") to ~~discarded~~ discard. The value of $ECODE may also be replaced via the SET command.

In section 6.3.2 Error Processing, section 'a)' replace the sentence:

> The value of $ECODE changes from an empty string to some other value as the result of an error or a SET command.

with

> 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 command.

In section 7.1.4.10,[2] for $EC[ODE] delete the sentence "This information is loaded by the implementation after detecting an erroneous condition, or by the application via the SET command."

In section 6.3.2 Error processing ~~delete the entire subsection beginning with: 'When $STACK($STACK,"ECODE") returns a non-empty string...' and the sub-paragraphs a) and b).~~ Replace the entire subsection beginning with: 'When $STACK($STACK,"ECODE") returns a non-empty string...' and the sub-paragraphs a) and b) with:

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

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

b) ~~It transfers control to the following line of code; this line is~~ **The following commands are** implicitly incorporated into the current execution environment immediately preceding the next command in the normal execution sequence:

~~ie (111)~~ TROLLBACK:$TLEVEL QUIT:$QUIT "" QUIT ; ~~eoi~~

N2.1.11 and N2.1.12 and N2.1.13

The ambiguity is resolved by changes due to N2.1.9 and N2.1.10.

N2.1.14

In section 6.3.2 Error Processing replace the paragraph " An Error Processing transfer of control consists of an implicit GOTO..." with:

> An Error Processing transfer of control consists of terminating the current command and processing in the scope of any active FOR

commands and indirection's. Execution **explicitly** ~~implicitly~~ resumes **at the same LEVEL with** ~~the following two lines~~ **two lines where the text of the first list is the value of $ETRAP and the second line is:**

QUIT:$QUIT "" QUIT

ls [li] *x* ~~$ETRAP V linebody~~ eol

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

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

4. Implementation Effects

4.1 Effect on Existing User Practices and Investments

There shouldn't be any.

4.2 Effect on Existing Vendor Practices and Investments

This should just bring the standard into agreement with existing practice. Both Micronetics and Intersystems have reviewed this and have not reported that it would cause and problems.

4.3 Techniques and Costs for Compliance Verification

No change from before.

4.4 Legal Considerations

5. Closely Related Standards Activities

5.1 Other X11 Proposals Under Consideration

Other proposals for which there are dependencies are listed in Section 1.5.

5.2 Other Related Standards Efforts

5.3 Recommendations for Coordinating Liaison

6. Associated Documents

7. Issues, Pros and Cons, and Discussion

May 1995 X11/SC15/ This document proposed for Type C approval.

Pro	Con
1 Corrects errors, ambiguities and inconsistencies in the X11.1 Canvas document [4]	1 Unspecified vendor impact [3]. See 4.2
	2 N2.1.14 unclear [7]
	3. Discussion buried in formalization
2 Has been implemented [2]	

Mar. 1997 X11/SC15/TG17/97-1

	Pro	Con
	1 Clarifies standard	
	2 Has been implemented	
	3.Long Awaited	

.Sept. 1997 X11/SC15/TG17/97-5

	Pro	Con
	1. Corrects error, Removes ambiguities, etc	none
	2. Has been implemented.	

.March. 1998 X11/SC15/TG17/98-2

	Pro	Con
	1. Clarifies Standard	none

8. Glossary

9. Appendix

6.3.2 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.

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

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

SESTACK counts stack levels since $ESTACK was last NEWed.

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

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

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"). If appending to $ECODE or $STACK($STACK,"ECODE") would exceed an implementations maximum string length, the implementation may choose which older information in $ECODE or $STACK($STACK,"ECODE") to discarded. 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 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) and a new error event 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:

~~ls {li}~~ TROLLBACK:$TLEVEL QUIT:$QUIT "" QUIT ; ~~eol~~

7.1.4.10 $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.

$ESTACK

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).

$ETRAP

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.

7.1.5.19 $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.

Values of intexpr less than -1 are reserved for future extensions of the $STACK function.

$ST[ACK] (intexpr , stackcodexpr)

```
stackcodexpr ::= expr V stackcode

                 PLACE
stackcode    ::= MCODE
                 ECODE
```

This form returns information about the action that created the level of the PROCESS-STACK identified by intexpr as follows:

intexpr 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 ] [ ^ | environment | 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 of the $STACK function. stackcodes differing only in the use of corresponding upper and lower case letters are equivalent.

8.2.18 SET

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 format required in section 7.1.4.10 for $ECODE, the SET of $ECODE to the value of the expr is not performed. An "M101" error is generated instead.
2) If the value of expr does conform to format required in section 7.1.4.10 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 which is updating $ECODE.

d) The value of $STACK($STACK,"MCODE") is replaced to reflect the SET command which 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.