19.3 Extended Error Message Processor

A. Purpose

The subroutines described here print error messages and diagnostic messages. These routines are intended primarily for use by other library routines.

B. Usage

B.1 Program Prototype, Setting or Getting a Message Processor Parameter

INTEGER MACT($\geq k_1$)**, IDAT**($\geq k_2$)

CHARACTER*($\geq k_3$**) TEXT**($\geq k_4$)

The values for the k_i's depend on the actions desired. For the actions described prior to the section on advanced features $k_1 = 2 \times$ (the number of actions) + 1, $k_2 = 1$, and k_3 and k_4 are 1 unless one of the actions is MEHEAD, in which case k_3 should be 4.

Assign values to MACT(). Prior to the section on advanced features, IDAT(), and TEXT() (with one exception) are not referenced.

CALL MESS(MACT, TEXT, IDAT)

B.2 Argument Definitions

MACT() [inout] Used to set or check parameters used by the message processor. Starting at MACT(1), is a list of integer pairs followed by a single MERET=51, which terminates the list. If the first integer is > 0, it is an index specifying the parameter that one wants to set and the following integer gives the value to be assigned to the parameter. If the first integer is negative, its absolute value specifies a parameter as above, and the value of that parameter is returned in the second integer. A single call can get and/or set as many of these parameters as desired. Below is a list of parameter names (which we recommend be used for the clarity of the code), and their values, that are used as the first integer in an entry. The second integer of the entry is denoted by Kj, where j is the value of the first integer.

MESUNI=10 (0 ≤ K10 ≤ 99) Set the scratch file unit number to K10. If the scratch unit is required and K10 has not been defined, its default value is $31 - k$, where k is the smallest positive integer for which unit $31 - k$ is available. If K10 is set to 0, a scratch unit is assumed not to be available, and tables with long lines will be printed with each row on multiple lines.

MEHEAD=11 (0 ≤ K11 ≤ 1) Defines the print that surrounds an error message. K11=0 gives nothing, and 1 gives TEXT(1)(1:4) repeated 18 times. If this is not used, one gets 72 $'s. (To get a blank line use 1 with TEXT(1) = ' '.)

MEDDIG=12 (-50 ≤ K12 ≤ 50) Set default digits to print for floating point. If K12 > 0 then K12 significant digits will be printed, if K12 < 0, then −K12 digits will be printed after the decimal point, and if K12 = 0, the default will be used, which is the full machine precision. Setting or getting this value will only work properly if the action is taken by calling SMESS or DMESS as appropriate; see below. There are separate internal values for K12 in SMESS and DMESS.

MEMLIN=13 (39 ≤ K13 ≤ 500) Set line length for diagnostic messages to K13. (Default is 128.)

MEELIN=14 (39 ≤ K14 ≤ 500) Set line length for error messages to K14. (Default is 79.)

MEMUNI=15 (-99 ≤ K15 ≤ 99) Set unit number for diagnostic messages to K15. If K15 = 0 (default), a Fortran PRINT statement is used.

MEEUNI=16 (-99 ≤ K16 ≤ 99) Set unit number for error messages to K16. If K16 = 0 (default), a Fortran PRINT statement is used.

MESCRN=17 (0 ≤ K17 ≤ 100000000) Set number of lines to print to standard output before pausing for "go" from user. Default is 0, which never stops.

MEDIAG=18 (0 ≤ K18 ≤ 1000000000) Not currently in use by MATH77 routines. Described more fully below.

MEMAXE=19 (0 ≤ K19 ≤ 1000000000) Set the maximum error value. When retrieving this value, it is the maximum value seen for $10000s + 1000p + i$, where s, p, and i are the stop and print levels, and the index on the last error message processed, respectively.

MESTOP=20 (0 ≤ K20 ≤ 8) Set the stop level for error messages. If an error message has a stop index > min(K20, 8), the program is stopped after processing the message. The default value is K20=3.

MEPRNT=21 (0 ≤ K21 ≤ 8) Set the print level for error messages. If an error message has a print index > K21, or the message is going to stop when finished, information in an error message is processed, else all the actions including

printing are skipped. (MESTOP controls stopping.) The default value is K21=3.

MERET=51 End of the list. (Only requires a single integer.)

TEXT() [in] Only referenced if MEHEAD is one of the actions. See the description above.

IDAT() [in] Not referenced by the application discussed here.

B.3 Getting and Setting the Default for Digits to Print for Floating Point Numbers

As mentioned above, the setting or retrieving of the number of decimal digits, MEDDIG above, requires a call to SMESS for single precision, and a call to DMESS for double precision. Either of these calls can be used to set any of the other parameters also. These calls require the additional declaration of a floating point array FDAT(), which must be single precision if SMESS is called and must be double precision if DMESS is called.

```
CALL SMESS(MACT, TEXT, IDAT, FDAT)
```

```
CALL DMESS(MACT, TEXT, IDAT, FDAT)
```

As above, the only argument actually used for this application is MACT().

B.4 Advanced Features

Most of the features listed below are used someplace in one of the library routines. The options that use FDAT require calling SMESS or DMESS, the other options can call MESS, SMESS, or DMESS. All variables passed to these routines are arrays which must have a declared dimension large enough to process the options being specified. Internal variables, all of which have a default value of 1, are used to keep track of locations as follows:

NTEXT	The next text to be output starts at TEXT(NTEXT).
NIDAT	The next output from IDAT starts at IDAT(NIDAT).
NFDAT	The next output from FDAT starts at FDAT(NFDAT).
NMDAT	The next output from MDAT starts at MDAT(NMDAT), where MDAT is defined by actions MEMDA1–MEMDA5 below, and NMDAT is set to one at the end of every text output.

An action that uses data pointed to by one of the above will cause the pointer to be incremented to one past the last location used. An exception is NMDAT, which when it reaches 5 is not incremented and the value pointed to is incremented instead.

When an option requires more than a single additional number to define it, a new letter followed by the index associated with the option is used to denote each additional number. The additional values of MACT available are:

MEDIAG=18 ($0 \le$ K18 ≤ 1000000000) Set the diagnostic level desired. Once again, note that this is not used in MATH77. The error message routines makes no use of K18. MESS merely serves as a place to set it and to answer inquiries on its value. It is intended to be set by users of library software. Library packages that make use of this number are expected to use it as described below. If K18 = 0 (the default), no diagnostic print is being requested. Else m = mod(K18, 256) determines whether a package will do diagnostic printing. Associated with a library package is a number L that must be a power of 2 < 129, and that should be mentioned in the documentation for the package. If the bit logical or(m,L) = L then diagnostic output for the routine with the associated value of L is activated. The value of L should have been selected by the following somewhat vague rules. Let $\log_2(\mathrm{L}) = 2i + j$, where j is 0 or 1. Select i = level of the library package, where the level is 0 if no other library routine that is likely to be used with the package could reasonably be expected to want any embedded diagnostics, and otherwise is min(4, I+1), where I is the maximum level for any library routine that is likely to be used with the package. Select $j = 0$ if the user is relatively unlikely to want diagnostics, and j = 1, if this is a routine for which considering its level the user is relatively likely to want diagnostic output. The next 8 bits, mod(K18/256, 256), may be used by the library routine to select the actual output that is to be given. These bits may be ignored, but if they are used, the lowest order bits should correspond to less voluminous output that is more likely to be requested. Finally, K18 / (2^{16}) may be used to give a count on how many times to print the diagnostics that are given. This count may be interpreted by library routines in slightly different ways, but when used it should serve to turn off all output after a certain limit is reached. By convention, if this is 0 there is no upper bound on the count.

METDIG=22 ($-50 \le$ K22 ≤ 50) As for MEDDIG (=12, above), except the value here is temporary, lasting until the return, or next use of this action. If 0, the internal value for K12 is used instead.

MENTXT=23 ($1 \le$ K23 ≤ 10000000) Set value of NTEXT to K23.

MEIDAT=24 ($1 \le$ K24 ≤ 1000000000) Set value of NIDAT to K24.

MEFDAT=25 ($1 \le$ K25 ≤ 1000000000) Set value of NFDAT to K25.

MEMDAT=26 ($1 \le$ K26 ≤ 5) Set value of NMDAT to K26.

MEMDA1=27 (K27) set MDAT(1) to K27. See description of NMDAT above.

MEMDA2=28 (K28) set MDAT(2) to K28.

MEMDA3=29 (K29) set MDAT(3) to K29.

MEMDA4=30 (K30) set MDAT(4) to K30.

MEMDA5=31 (K31) set MDAT(5) to K31.

METABS=32 ($1 \le$ K32 ≤ 100) set spacing for tabs to K32. The default value is K32=6.

MEERRS=33 (K33) set the current error counter to K33. (To retrieve this value set a location in MACT() to -33, and the count will be returned in the following location.) This count is set to 0 on the first call made to MESS.

MECONT=50 Exit, but no print of current print buffer. The error or diagnostic message is to be continued immediately.

MERET=51 All done with diagnostic or error message. Complete processing and return; or for some error messages stop.

MEEMES=52 (K52, L52, M52) Start an error message with severity level K52, index for the error of L52, and message text starting at TEXT(M52). If M52 is 0, message text starts at TEXT(NTEXT), and if M52 < 0, no message text is printed as part of this action. This option assumes that TEXT(1) starts with text giving the name of the subprogram or package terminated with a '$B'. Library routines should set K52 $= 10 \times s + p$, where s is the stop level desired, and p the print level, and should have $10 > p \ge s \ge 0$. We offer the following guidelines as a yardstick for setting the value of s.

- **= 9** User has ignored warning that program was going to be stopped.
- **= 8** Program has no way to continue.
- **= 7** User has given no indication of knowing that functionality of results is reduced. (E.g. not enough space for some result.)
- **= 6** Program could continue but with reduced functionality.
- **= 5** Results far worse than user expected to want.
- **= 4** User has given no indication of knowing that results do not meet requested or expected accuracy.
- **= 3** Warning is given that program will be stopped without some kind of response from the calling program.
- **= 2** Program is not delivering requested or expected accuracy.
- **= 1** Some kind of problem that user could correct with more care in coding or in problem formulation.
- **= 0** Message is for information of uncritical nature.

Print levels might be counted down so that warnings given several times are no longer given, or be counted up so that a warning is only given after a certain threshold is reached. Levels should be selected with the understanding that the default is to print or stop only for levels > 3.

METEXT=53 Print TEXT, starting at TEXT(NTEXT). Print ends with the last character preceding the first '$'. Special actions are determined by the character following the '$'. Except as noted, the '$' and the single character that follows are not printed. In the text below, "to continue", means to continue print of TEXT with the next character until the next "$". Except for the one case noted, NTEXT is set to point to the second character after the "$". Possibilities for the character following the "$" are (letters must be in upper case):

- **B** Break text, but don't start a new line.
- **E** End of text and line.
- **R** Break text, don't change the value of NTEXT. Thus next text Repeats the current.
- **N** Start a New line, and continue.
- **I** Print IDAT(NIDAT), set NIDAT = NIDAT + 1, and continue.
- **J** As for I above, except use the last integer format defined by a "$(", see below.
- **F** Print FDAT(NFDAT), set NFDAT = NFDAT + 1, and continue.
- **G** As for F above, except use the last floating format defined by a "$(", see below.
- **M** Print MDAT(NMDAT), set NMDAT = NMDAT + 1, and continue.
- **H** Marks terminator for column and row Headings; see table, vector, and matrix output below. This causes enough blanks to be generated to keep column headings centered over their columns. After the blanks are generated, text is continued until the next '$'. This is not to be used except inside column or row headings. The last row or column should be terminated with a '$E' or if appropriate, a '$#' for a row or column label.
- **(** Starts the definition of a format for integer or floating point output. The format must require no more than 12 characters for floating point and may not contain a "P" field (e.g.

"(nnEww.ddEe)", where each of the lower case letters represents a single digit), and no more than 7 characters for integer output. If the next character following the ")" that ends the format is not a "$" then "$J" or "$G" type output is done; see above. In either case processing of TEXT then continues.

T Tab. See METABS (=32 above).

Used in matrix row or column labels. This prints the current row or column index, respectively, ends the text for the current row or column, and resets the text pointer to where it started.

**$** a single '$' is printed; continue output of text.

− Starts a negative number for skipping, see next below.

0-9 A sequence of digits (perhaps preceded by a '−' sign defines an extra amount to skip the index (ahead or back) for fetching the next thing from FDAT or IDAT (whichever is referenced next). We recommend this be used just prior to the $ item that requires the skip. (The default is to start one past the last thing printed.)

C Only used by `pmess` which deletes it and the preceding '$'. Used at the end of a line to indicate Continued text.

blank A blank ending up in column 72 is replaced by "$ " by pmess, thus avoiding bugs in some Fortran compilers. If the user inputs a "$ ", not followed by anything pmess starts a new data statement with an incremented array name; if followed by a "D" pmess will arrange for the preceding text to be stored in an array (so that not too many continuation lines are required) and outputs a comment so the C conversion is done correctly.

other Don't use this — the '$' is ignored, but new features may change the action.

ME????=54 Not used.

METABL=55 (K55, L55, M55, N55) Note this action automatically returns when done, further locations in MACT are not examined. This action prints a heading and/or data that follows a heading. If K55 is 1, then the heading text starting in TEXT(NTEXT) is printed prior to printing the data. This text should contain embedded "$H"'s to terminate columns of the heading. If there is no heading on a column, use " $H". Note the leading blank. If the heading is to continue over k columns, begin the text with "$H" repeated $k-1$ times with no other embedded characters. The very last column must be terminated with "$E" rather than "$H". After tabular data are printed, K55 is incremented by 1, and compared with L55. If K55 > L55, K55 is reset to 1, and if the data that was to be printed required lines that were too long, data saved in the scratch file is printed using the headings for the columns that would not fit on the first pass. Note that only one line of tabular data can be printed on one call to this subroutine.
M55 gives the number of columns of data associated with the heading which is the sum of the "rr" values. N55 is a vector containing as many entries as needed to get the sum of the "rr" values equal to M55. The k^{th} integer in N55 defines the printing action for the k^{th} column of the table. Let such an integer have a value defined by $rr+100\times(t+10\times(dd+100\times ww))$, i.e. $wwddtrr$, where $0 \le rr, dd, ww < 100$, and $0 \le t < 10$.

rr The number of items to print.

t The type of output.

- **1** Print text starting at TEXT(NTEXT), rr = 01.
- **2** Print the value of K55, rr = 01.
- **3** Print integers starting at IDAT(NIDAT).
- **4** Print starting at FDAT(NFDAT), using an F format.
- **5** Print starting at FDAT(NFDAT), using an E format.
- **6** Print starting at FDAT(NFDAT), using an G format.

dd Number of digits after the decimal point.

ww The total number of print positions used by the column, including the space used to separate this column from the preceding one. This must be big enough so that the column headings will fit without overlap.

MEIVEC=57 (K57) Print IDAT as a vector with K57 entries. The vector output starts on the current line even if the current line contains text. This is useful for labeling the vector. The vector starts at IDAT(NIDAT). If K57 < 0, indexes printed in the labels for the vector start at NIDAT, and entries from NIDAT to −K57 are printed.

MEIMAT=58 (K58, L58, M58, I58, J58) Print IDAT as a matrix with K58 declared rows, L58 actual rows, and M58 columns. If K58 < 0, instead of using 1 for the increment between rows, and K58 for the increment between columns, −K58 is used for the increment between rows, and 1 is used for the increment between columns. If L58 < 0, the number of actual rows is mod(−L58, 100000), and the starting row index is −L58 / 100000. Similarly for M58 < 0. TEXT(I58) starts the text for printing row labels. If I58 < 0, no row labels are printed. If I58 = 0, it is as if it pointed to text containing "Row $E". Any

"$" in a row or column label must be followed by "H" or "E" which terminates the text for the label. In the case of $H, text for the next label follows immediately, in the case of $E the current row index is printed in place of the $E and the next label uses the same text. J58 is treated similarly to I58, except for column labels, and with "Row $E" replaced with "Col $E". The matrix starts at IDAT(NIDAT), and NIDAT points one past the end of the matrix when finished.

MEJVEC=59 (K59) As for MEIVEC, except use the format set from using $(.

MEJMAT=60 (K60, L60, M60, I60, J60) As for MEIMAT, except use the format set from using $(.

MEFVEC=61 (K61) As for MEIVEC, except print FDAT as a vector with K61 entries. The vector starts at FDAT(NFDAT).

MEFMAT=62 (K62, L62, M62, I62, J62) As for action MEIMAT, but print FDAT instead, and use NFDAT in place of NIDAT.

MEGVEC=63 (K63) As for MEFVEC, except use the format set by using $(.

MEGMAT=64 (K64, L64, M64, I64, J64) As for MEIMAT, except use the format set by using $(.

MEIVCI=65 (K65, L65) As for MEIVEC, except the vector entries have a spacing of K65, and there are L65 entries in the vector.

MEJVCI=66 (K66) As for MEIVCI, except use the format set by using $(.

MEFVCI=67 (K67, L67) As for MEFVEC, except the vector entries have a spacing of K67, and there are L67 entries in the vector.

MEGVCI=68 (K68) As for MEFVCI, except use the format set by using $(.

MEFSPV=69 (K59) K59 gives the number of entries in the sparse vector, IDAT gives indexes of the entries, and FDAT gives the corresponding values. One may want to make use of MEMDA1, and MEMDA2 to save for example the column index and the number of entries for output in a text message that precedes the output of the vector. The user will need to write a loop to output a sparse matrix, and output of sparse integer vectors in not supported.

B.5 Constructing Message Data with `pmess`

For complicated messages, making up the Fortran DATA statements, required can be quite difficult. Even for simple cases, we recommend using the stand alone program `pmess` to construct both the DATA statements, and PARAMETER statements that define variables that can be used to reference messages. One can then change input message text, run `pmess`, and replace the old output in the code with the new output, without requiring other changes in the code. `pmess` expects input from "standard input", and writes output to the file `tmess`. Thus for the example given in the section below, with an input file of `drsmess.err` the program is executed as follows (at least for DOS and Unix)

```
pmess <drsmess.err
```

and the file `tmess` can be inserted into a Fortran routine. In order to follow the declaration order imposed by the Fortran standard, and to avoid splitting the output generated by `pmess`, we recommend inserting the output from `pmess` after all other specification statements in the code, but before other data statements (if any).

`pmess` generates parameters LTXTAA, LTXTAB, ... These names are associated with individual messages that might be printed, and are identified by comments of the form cAA, cAB, ... generated by `pmess`. The values generated for these parameters are the character positions of the message in the particular character array in which that message is stored. Messages are first stored in a character array named MTXTAA, but if a line consisting of a single '$' in column 1 is encountered the last two letters of this name are advanced as for LTXTxx, and further text is stored in the new character array. `pmess` generates both the declarations and the data for these character arrays.

The comments (and just the comments) generated by `pmess` can be used as inputs to `pmess`. Thus we recommend including these comments in the code both as documentation for the messages, and to simplify generating the data if changes to messages are desired. If these comments are passed as input to `pmess`, then the reference to column 1 above should be to column 5.

`pmess` replaces a blank in column 72 of output character strings with a '$', and inserts a blank in the continued string. This is done because some editors delete trailing blanks, and some compilers require that the trailing blanks in continued character data be present.

There are three special cases used in `pmess`, none of which generates text in the output arrays. A "$C" at the end of a line indicates that the text on the following line is a continuation of the text on this line. At least for use in a Fortran 77 environment one should not use input lines longer than 68 characters to allow for the 4 extra spaces required by the generated comments. A "$ " at the start of a line causes the code to use a new array for the following text, and a "$ D" at the start of a line tells `pmess` to separate the data for the preceding text into separate array elements. The latter is necessary if the text would require more than 19 continuation lines.

Special comments for the conversion to C are also output for this case.

Recall that for error messages, the text must start with the subprogram name, this name should be terminated with '$B'. We suggest that after the subprogram name, one give the text for the error messages in order. Then the text for error message i, can be located from an array referenced by the error index.

C. Examples and Remarks

`pmess` transformed the data in `drsmess.err` to statements included in the listing for DRSMESS.FOR. The output from running DRSMESS is given in ODSMESS. This illustrates how error messages can be generated, and how the length of the output line can be changed from the default (which is 79) to 40.

Other library routines can be examined to see how various error messages and diagnostic outputs have been implemented. Most of the features described here can be found in the source code for either DIVA or DIVADB,

D. Functional Description

The routines mentioned here are called by a number of library routines for the printing of both error messages and diagnostic information. The user can call the routines described here to alter the actions they take when called by other library routines.

Other approaches to handling error messages can be found in the references cited below.

References

1. P. A. Fox, A. D. Hall, and N. L. Schryer, *The PORT mathematical subroutine library*, **ACM Trans. on Math. Software 4**, 2 (June 1978) 104–126.

2. P. A. Fox, A. D. Hall, and N. L. Schryer, *Algorithm 528: Framework for a portable library [Z]*, **ACM Trans. on Math. Software 4**, 2 (June 1978) 177–188.

3. IMSL, **MATH/LIBRARY Version 1.1**, 2500 ParkWest Tower One, 2500 CityWest Boulevard, Houston TX 77042-3020 (1989) 1130–1136.

4. Rondall E. Jones and David K. Kahaner, *XERROR, the SLATEC error-handling package*, **Software Practice and Experience 13** (1983) 251–257.

E. Error Procedures and Restrictions

Use of indexes other than those described here, cause MESS to print the message "Actions in MESS terminated due to error in usage of MESS." and then to return. There are a two errors that cause MESS to execute an unrequested "STOP." Such a STOP will print one of the two following messages.

"Could not assign scratch unit in MESS."
"Stopped in MESS -- Column width too small in a heading."

F. Supporting Information

The source language is ANSI Fortran 77.

Algorithm and code due to F. T. Krogh, JPL, November, 1991. Last change in March, 2006 to add output of sparse vectors.

Entry	Required Files
DMESS	AMACH, DMESS, MESS
MESS	AMACH, MESS
SMESS	AMACH, MESS, SMESS

drsmess.err

```
DRSMESS$B
 Description of error.  (With one real)  FDATA1 = $F.$E
 Description of 2nd error.  (With no data)$E
 Description of 3rd error.  (With one integer)  IDATA1 = $I.$E
 Description of 4th error.  (With two integers and one real)$N
IDATA1 = $I, IDATA2 = $I, FDATA2 = $F.$E
```

DRSMESS

```
c      program DRSMESS
c>> 1998-11-02 DRSMESS Krogh  Typed all variables.
c>> 1994-09-09 DRSMESS Krogh  Added CHGTYP code.
c>> 1993-06-25 DRSMESS Krogh  Additions for Conversion to C.
c>> 1992-03-24 DRSMESS Krogh -- Initial Code.
c
c--S replaces "?": DR?MESS, ?MESS
c
      integer           I, IDAT(3), ILOC(4), MACT(5), MACT1(3)
      real              FDAT(2)
      logical           LIN40
c
      integer MERET, MEEMES, MESTOP, MEELIN
      parameter (MERET  =51)
      parameter (MEEMES =52)
      parameter (MESTOP =20)
      parameter (MEELIN =14)
      integer MLOC(4)
c ********* Error message text ***************
c[Last 2 letters of Param. name]  [Text generating message.]
cAA DRSMESS$B
cAB Description of error.  (With one real)  FDATA1 = $F.$E
cAC Description of 2nd error.  (With no data)$E
cAD Description of 3rd error.  (With one integer)  IDATA1 = $I.$E
cAE Description of 4th error.  (With two integers and one real)$N
c   IDATA1 = $I, IDATA2 = $I, FDATA2 = $F.$E
      integer LTXTAA,LTXTAB,LTXTAC,LTXTAD,LTXTAE
      parameter (LTXTAA=  1,LTXTAB= 10,LTXTAC= 64,LTXTAD=107,LTXTAE=168)
      character MTXTAA(2) * (135)
      data MTXTAA/'DRSMESS$BDescription of error.  (With one real)  FDAT
     *A1 = $F.$EDescription of 2nd error.  (With no data)$EDescription o
     *f 3rd error.  (W','ith one integer)  IDATA1 = $I.$EDescription of$
     * 4th error.  (With two integers and one real)$NIDATA1 = $I, IDATA2
     * = $I, FDATA2 = $F.$E '/
c  ---------- End of code generated by PMESS from DRSMESS.ERR ----------
      data MLOC /LTXTAB,LTXTAC,LTXTAD,LTXTAE/
c
c                   1       2 3 4       5
      data MACT / MEEMES,25,0,0, MERET /
      data MACT1 / MEELIN, 40, MERET /
      data FDAT / 1.7E-12, -12.3456789E0 /
      data IDAT / 17, -178, 4 /
      data ILOC / 1, 1, 2, 2 /
c
      LIN40 = .false.
c                         Loop to print error messages.
   10 do 100 I = 1, 4
         MACT(3) = I
         MACT(4) = MLOC(I)
         call SMESS(MACT, MTXTAA, IDAT(ILOC(I)), FDAT(ILOC(I)))
  100 continue
      if (LIN40) stop
c                         Change the line length to 40 and do it again.
      call MESS(MACT1, MTXTAA, IDAT)
      LIN40 = .true.
      go to 10
      end
```

ODSMESS

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2, Print level = 5, Error index = 1
Description of error.  (With one real)  FDATA1 = 1.6999999E-12.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2, Print level = 5, Error index = 2
Description of 2nd error.  (With no data)
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2, Print level = 5, Error index = 3
Description of 3rd error.  (With one integer)  IDATA1 = -178.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2, Print level = 5, Error index = 4
Description of 4th error.  (With two integers and one real)
IDATA1 = -178, IDATA2 = 4, FDATA2 = -12.345679.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2,
Print level = 5, Error index = 1
Description of error.  (With one real)
FDATA1 = 1.6999999E-12.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2,
Print level = 5, Error index = 2
Description of 2nd error.  (With no
data)
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2,
Print level = 5, Error index = 3
Description of 3rd error.  (With one
integer)  IDATA1 = -178.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
DRSMESS reports error: Stop level = 2,
Print level = 5, Error index = 4
Description of 4th error.  (With two
integers and one real)
IDATA1 = -178, IDATA2 = 4, FDATA2 =
-12.345679.
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```