UMTRI-83-19

[illegible]

ADAAS
AUTOMATED DATA ACCESS AND ANALYSIS SYSTEM

SUBROUTINE DOCUMENTATION MANUAL

John A. Green

MAY 1983

UMTRI The University of Michigan
Transportation Research Institute

Technical Report Documentation Page

1. Report No. UMTRI-83-19	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ADAAS Automated Data Access and Analysis System Subroutine Documentation Manual		5. Report Date May 1983 6. Performing Organization Code
7. Author(s)		8. Performing Organization Report No. UMTRI-83-19
9. Performing Organization Name and Address Transportation Research Institute The University of Michigan 2901 Baxter Rd. Ann Arbor, MI 48109		10. Work Unit No. (TRAIS) 11. Contract or Grant No. 1133
12. Sponsoring Agency Name and Address Motor Vehicle Manufacturers Association 320 New Center Building Detroit, MI 48202		13. Type of Report and Period Covered 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This manual documents a library of computer subroutines that were developed by the University of Michigan Transportation Research Institute in support of the data base efforts of the Institute's Data Center. The subroutines are written in FORTRAN and 370 Assembler.

Four classes of routines are included:

1) Data handling/character manipulation,
2) Support for interactive programs,
3) Dictionary processing, and
4) ADAAS program support.

17. Key Words	18. Distribution Statement

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 150	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Report Number UMTRI-83-19

A D A A S

Automated Data Access and Analysis System

Subroutine Documentation Manual

by

John A. Green

May 1983

The University of Michigan
Institute of Science and Technology
Transportation Research Institute
Ann Arbor, Michigan

ACKNOWLEDGEMENTS

The subroutines documented in this manual are the result of contributions from many programmers over a long period of time. Many of the important routines have been re-written many times, and consequently represent the best thoughts of a number of people.

The primary people responsible for the present library are: Cory Devor, John Ferguson, Hank Golomb, John Green, Carole Hafner, Elliot Noma, and Marianne Stover.

The dynamic dimensioning routine DIME was written by the Statistical Research Laboratory at the University of Michigan. The version documented here is an early version of the routine currently maintained in STAT:LIBRARY. Permission to use this routine is appreciated.

Primary funding for the development and maintenance of this program library has come from the Motor Vehicle Manufacturers Association. The continuing support of that agency is gratefully acknowledged.

TABLE OF CONTENTS

INTRODUCTION

This manual documents a library of subroutines developed by the Transportation Research Institute at the University of Michigan in support of the data base efforts of the Institute's Transportation Data Center. The subroutines are written in FORTRAN or 370 Assembler and are stored in the file HSRI:LIBRARY. To include these subroutines with a compiled program (called MYPROG) use the MTS $RUN command

```
$RUN MYPROG+HSRI:LIBRARY
```

or include the statement

```
$CONTINUE WITH HSRI:LIBRARY
```

as the last line in MYPROG.

The library contains four general groups of subroutines:

1) Character manipulation, or data handling routines for use in data formatting applications.

2) Support routines for interactive user-oriented programs (keyword scanners, command scanners, list processors, etc.)

3) Routines for modifying, listing, and processing OSIRIS type 1 and type 5 dictionaries.

4) Data access routines for the ADAAS program. Due to the specific requirements of these routines, they are of limited use in a general programming environment.

The routines in the library are listed below by this classification for easy identification.

1) Data processing

CHSRT	- Sort small arrays
FILLM	- Perform multiple fill operations
ITRNSL	- Translate character strings
JULDAT	- Convert gregorian dates to julian
MOVBUT	- Move numeric characters
MOVEM	- Perform multiple move operations

2) Program support functions

CMDSCN	- Command interpreter
DIME	- Run-time array dimensioning
DISTIM	- Timer interrupt
EKOLIN	- Print output with prefix & line wrap

ESCAPE	Attention interrupt processing
EWRITE	WRITE routine with error handling
FAIL	Program interrupt processing
FWRT	Covert real numbers to character
GETTP	Run-time access to tapes & files
GUSRIN	Read input from GUSER
INFILE	Input file control routines
INFOF	File type
IWRT	Convert integer numbers to character
KEYSCN	Keyword/Modifier interpreter
LEFJ	Left justify and delete blanks
LNBTD	Covert MTS line numbers to character
LSTFIX	Sort number lists & delete duplicates
OUTFIL	Output file control routines
PDNCHK	Check pseudo-device names
PRNTCK	Check strings for printing characters
QSAM	Fixed-block (FB) read/write routines
RBTD	Convert integer with decimal places
READIN	Read input from SCARDS
SHFTST	Shift a substring right or left
SLIST	List interpreter
TIMDAT	Time and date string generator
VLIST	Number list interpreter
VRANGE	Value+range interpreter

In addition, there are three special purpose routines that support CMDSCN and KEYSCN
LSTPAR REPMSG SPLCHK

3) Dictionary processing

DIC1T5	Convert type 1 to type 5
DIC5T1	Convert type 5 to type 1
FIVPAR	Convert type 5 parameters to binary
LISFIV	List type 5 records

4) ADAAS routines

APIN	Read, filter, and recode data
CHKVAR	Check variable numbers
DICPAR	Get dictionary information
GETDAT	Access input data set
IFILTR	Filter/Recode interpreter
ILABEL	Label program output
FLOT	Float integer numbers
VLCHEK	Check variable lists

SUBROUTINE DOCUMENTATION

Module Name: APIN
(Analysis Program Input)

Purpose:

To read records from the input data set, perform filter or recode operations requested, and return the requested data to the calling program. There are five entry points:

SETFIL	- To record the filter parameters from IFILTR
SETREC	- To record the recode parameters from IFILTR
SETVAR	- Read data set dictionary and recode program variables.
IRECHK	- To determine if a given variable has been recoded.
CASE	- To return a valid case (i.e., filtered & recoded).

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Subroutines Used:

DICPAR, FLOT, GETDAT, GETSPACE, INFILE, SERCOM

Logical I/O Units:

SERCOM - Error messages

Description:

The subroutine IFILTR should be called first to read and decode filter and recode statements. IFILTR in turn calls SETFIL and SETREC to store the decoded filter and recode information. Analysis programs then pass a list of required program variables by means of the SETVAR entry. SETVAR interrogates the on-line dictionary by means of GETDAT and DICPAR and stores all the variables parameters internally and returns them for use in the analysis programs. The CASE entry may then be called to sequentially return records from the input data set that have been properly filtered, recoded, and floated, if required.

Entry Point:	SETFIL
Module Name:	APIN

Purpose:

To transfer filter parameters decoded by the input routine IFILTR to internal storage for later use by CASE.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL SETFIL(FILLST,NFVAR,ONOFF,FNRES,LINK,INEX,XIO,FLOC)
```

Parameters:

FILLST	The Integer*2 list of filter variables.
NFVAR	The Integer*4 number of filter variables.
ONOFF	Indicates ranges in the value list.
FNRES	Number of responses.
LINK	Indicates AND or OR logical connectives.
INEX	INCLUDE or EXCLUDE indicator.
XIO	Filter syntax vector.
FLOC	Record location.

<u>Entry Point:</u> SETREC

<u>Module Name:</u> APIN

<u>Purpose:</u>

To transfer recoded parameters decoded by the input routine IFILTR to internal storage for later use by CASE.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Calling Sequence:</u>

```
CALL SETREC(RECLST,NRVAR,ISYN,JRANGE,NVST,JRST,RNRES,RLOC)
```

<u>Parameters:</u>

RECLST	The Integer*2 list of recode variables.
NRVAR	The Integer*4 number of recode variables.
ISYN	Recode syntax vector.
JRANGE	Indicates ranges in the value list.
NVST	Pointers to ISYN.
JRST	Pointers to JRANGE.
RNRES	Number of responses.
RLOC	Record location.

Entry Point:	SETVAR
Module Name:	APIN

Purpose:

To read and store parameters for program variables, float the missing data codes if required, and acquire an input buffer.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL SETVAR(VARLST,NLVAR,CHRTYP,VARNAM,RECLOC,FLDWTH,
            NUMDEC,NUMRES,MISONE,MISTWO,MODE LSTTYP,
            LISTSW,OUTWTH,&RC4)
```

Parameters:

VARLST	The NLVAR halfword list of program variables.
NLVAR	The fullword number of program variables.
CHRTYP	The NLVAR halfword list of character types.
VARNAM	The 24*NLVAR list of variable names
RECLOC	The NLVAR halfword list of record locations.
FLDWTH	The NLVAR halfword list of field widths.
NUMDEC	The NLVAR halfword list of implied decimal places.
NUMRES	The NLVAR halfword list of number of responses.
MISONE	The NLVAR fullword list of missing data code #1.
MISTWO	The NLVAR fullword list of missing data code #2.
MODE	0 - Return data in integer (fullword) mode 1 - Return data in floating point mode. 2 - Return data in character numeric mode.
LSTTYP	Fullword type of variable list
LISTSW	Fullword dictionary list switch (see DICPAR)

Parameters: (Continued)

OUTWTH	Fullword length of the output field. In words for MODE = 0,1 In bytes for MODE = 2

Return Code(s):

&RC4 - Error return from GETDAT, GETSPACE, or DICPAR

Description:

The data set dictionary is read and parameters for the program variables in VARLST are stored in the appropriate arrays. If the output data format is floating point, all the missing data codes are floated.

Entry Point: IRECHK

Module Name: APIN

Purpose:

To determine if a specified variable number is in the recode variable list.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
INTEGER*4   IRECHK,IRETRN
   ...
IRETRN = IRECHK(VARNUM)
```

Parameters:

VARNUM The halfword variable number to be checked.

IRETRN 0 - The variable has not been recoded.
1 - The variable has been recoded.

Entry Point: CASE

Module Name: APIN

Purpose:

To read data records, perform the required filter and recode operations, float the data if required, and place the data in the output array for subsequent use by the analysis programs.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
CALL CASE(OUTPUT,&RC4,&RC8)
```

Parameters:

OUTPUT The output data region. In FORTRAN programs, OUTPUT should be declared as follows:

```
INTEGER*4 for MODE = 0
REAL*4    for MODE = 1
LOGICAL*1 for MODE = 2
```

Return Code(s):

&RC4 - End-of file on input data set.

&RC8 - Error return from GETCHA or CLOIN

Description:

The data set record is read into the input data region by GETCHA and the required filtering and recode operations are performed. The data is then moved into the output array in the mode specified by the SETVAR entry.

Entry Point: CHKVAR

Module Name: CHKVAR

Purpose:

To check if a specified variable number exists in the active data set, and optionally, if it is numeric.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
LOGICAL*1   CHEKSW,CHKVAR
   ...
CHEKSW = CHKVAR(NUMBER,MODE)
```

Parameters:

NUMBER	The INTEGER*2 variable number to be checked.
MODE	The INTEGER*4 mode of operation. 0 - Check if the variable exists. 1 - Check if the variable exists and is numeric.
CHEKSW	The LOGICAL*1 result of the check.

Subroutines Used: BTD, FIVPAR, IRECHK, MOVEC, READ, SERCOM

Logical I/O Units:

READ	Read records from on-line dictionary
SERCOM	Error messages

Description:

The on-line dictionary -ADASDICT is read to see if the specified variable exists. If numeric checking is specified (i.e., MODE = 1) then the character type is checked to see if it is numeric or if it is alphabetic but has been recoded.

Restriction:

References to an on-line dictionary and to several named common areas make this subroutine useful only in the ADAAS program.

Entry Point: CHSRT

Module Name: CHSRT

Purpose:

To sort small in-core arrays into ascending order.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL CHSRT(ARRAY,NUMA,LENA,LOCF,LENF,&RC4)

Parameters:

ARRAY	The LOGICAL*1 array containing the data to be sorted.
NUMA	The INTEGER*4 number of segments in ARRAY which are to be sorted.
LENA	The INTEGER*4 length of each segment in bytes. LENA must be less than or equal to 80.
LOCF	The INTEGER*4 location within LENA where the sort field begins.
LENF	The INTEGER*4 length of the sort field in bytes.

Return Code(s):

&RC4 - ERROR: CHSRT WIDTH > 80.

Subroutines Used: ICLC, MOVEC, SERCOM

Logical I/O Units:

SERCOM - Error message LENA > 80

Description:

The purpose of CHSRT is to sort elements of ARRAY into ascending order while the information is stored in core. The array ARRAY is treated as consisting of NUMA segments, each segment being LENA bytes long. Within each segment there is a field LENF bytes long that begins at LOCF. The segments of ARRAY are sorted into ascending order on the basis of the binary values of the field defined by LOCF and LENF.

For long lists and/or multiple sort fields involving input or output operations, the MTS *SORT facility should be used. CHSRT is intended for use where multiple short sorts may be needed and generation of the calling sequences for *SORT would be cumbersome.

Entry Point: CMDSCN

Module Name: CMDSCN

Purpose:

To scan a character string for the existence of a member of a predefined set of commands.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL CMDSCN(STRING,LENGTH,MAXLEN,NCMDS,COMMND,CMDPAR,
            CMDNUM,LAST,&RC4,&RC8)
```

Parameters:

STRING	A Logical*1 array of dimension MAXLEN that contains the character string to decode. The string must terminate with a trailing blank.
LENGTH	The Integer*4 length of the character string in STRING.
MAXLEN	The Integer*4 dimension of STRING.
NCMDS	The Integer*4 number of possible commands that are defined.
COMMND	A Logical*1 array of dimension 8*NCMDS containing the left-justified, 8-byte-aligned command names.
CMDPAR	An Integer*2 array of dimension (2,NCMDS). CMDPAR(1,J) is the minimum number of characters that will be recognized as an abbreviation for command number J. CMDPAR(2,J) is the full length of the name for command number J.
CMDNUM	The Integer*4 number of the command found in STRING on output. If the command is an MTS command, that is it begins with "$", then CMDNUM=0.
LAST	The Integer*4 location of the first blank in STRING following the command. This variable is provided for use with KEYSCN as a pointer to the start of a KEYWORD/MODIFIER string.

Return Code(s):

&RC4	Unrecognizable command in batch mode
	CANCELled command
	Invalid LENGTH
	NCMDS < 1
	Blank string, or no trailing blank

Subroutines Used:

EQUC, FINDC, FINDST, GUINFO, GUSRIN, IGC, LCOMC, LSTPAR, MOVEC, PRNTCK, REPMSG, SERCOM, SHFTST, SPLCHK

Logical I/O Units:

SERCOM	Error messages

Special Note:

A 140-byte COMMON area named /BUF/ is used for I/O operations and other temporary tasks.

Description:

The array STRING is scanned for a command that matches one in the input list defined by COMMND. If a command is present it must be the first non-blank word in STRING. If a syntax error is encountered in BATCH mode, an error message is printed and RC4 is taken. If a syntax error is encountered in TERMINAL mode, an attempt is made to determine if the invalid command is a possible misspelling of a correct one. The user is queried for verification or rejection of each possible misspelling that is found. If this process is unsuccessful, an error message is printed and the user is prompted for a replacement string or the word CANCEL to cancel the entire scanning process. If a replacement string is supplied, the incorrect item is deleted from STRING and the replacement is added at the beginning of the array providing that the array dimension is not exceeded. On exit from CMDSCN, therefore, STRING contains a corrected version of the input.

A space (X'40') is the normal break character separating the command from any modifiers and keywords that may occur. Leading blanks may also occur at any point.

Entry Point: DICPAR

Module Name: DICPAR

Purpose:

To retrieve the dictionary parameters for a list of variable numbers.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL DICPAR(VARLST,NLVAR,CHRTYP,VARNAM,RECLOC,FLDWTH,NUMDEC,
            NUMRES,MISONE,MISTWO,MODE,LSTTYP,LISTSW,OUTWTH,
            &RC4)
```

Parameters:

VARLST	The INTEGER*2 list of variable numbers.
NLVAR	The INTEGER*4 number of variables in VARLST for LSTTYP = 1
CHRTYP	The INTEGER*2 list character types returned.
VARNAM	The LOGICAL*1 (i.e., VARNAM(24,NLVAR) list of variable names returned.
RECLOC	The INTEGER*2 list of record locations returned.
FLDWTH	The INTEGER*2 list of field widths returned.
NUMDEC	The INTEGER*2 list of implied decimal places returned.
NUMRES	The INTEGER*2 list of number of responses returned.
MISONE	The INTEGER*4 list of MD code #1 returned.
MISTWO	The INTEGER*4 list of MD code #2 returned.
MODE	The INTEGER*4 data return mode as specified in the APIN entry SETVAR. 0 - Fullword integer binary 1 - Fullword floating point binary 2 - Character numeric

Parameters: (Continued)

LSTTYP	The INTEGER*4 type of variable list specified as input. 1 - List contained in VARLST 2 - ALLV (all variables in the dictionary) 3 - ALLNV (all numeric variables in the dictionary)
LISTSW	The INTEGER*4 dictionary list control switch. 0 - Don't list the dictionary 1 - List the dictionary on PPRNT.
OUTWTH	The INTEGER*4 total output record length for the requested variables (in WORDS for MODE = 0,1 or in BYTES for MODE = 2).

Subroutines Used:

BTD, EWRITE, FIVPAR, IRECHK, LISFIV, MOVEC, READ, SERCOM

Logical I/O Units:

PPRNT	Dictionary list for LISTSW = 1
READ	Read records from on-line dictionary
SERCOM	Error messages

Description:

The on-line dictionary (FDUB IDICT) is read to see if the specified variable exists. If numeric checking is specified (i.e., MODE = 1) then the character type is checked to see if it is numeric or if it is alphabetic but has been recoded. The parameters from the dictionary record are converted to binary and stored in the appropriate array.

Restriction:

References to an on-line dictionary and to several named common areas make this subroutine useful only in the ADAAS program.

Entry Point: DIC1T5

Module Name: DIC1T5

Purpose:

To convert OSIRIS type 1 dictionary records to type 5

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL DIC1T5(DICONE,DICFIV)
```

Parameters:

DICONE	The 80-character LOGICAL*1 type 1 dictionary record to be converted.
DICFIV	The 80-character LOGICAL*1 type 5 dictionary record.

Subroutines Used:

BTD, DTB, IGC, MOVEC, SETC

Entry Point: DIC5T1

Module Name: DIC5T1

Purpose:

To convert an OSIRIS type 5 dictionary record to type 1.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL DIC5T1(DICFIV,DICONE)
```

Parameters:

DICFIV	The 80-character LOGICAL*1 type 5 dictionary record to be converted.
DICONE	The 80-character LOGICAL*1 type 1 dictionary record.

Subroutines Used:

BTD, DTB, IGC, MOVEC, SETC

Module Name: DIME

Purpose:

To dynamically dimension FORTRAN arrays during program execution. There are four entry points for the routine:

DIME -	Allocates new arrays.
REDIME -	Changes the allocation of existing arrays.
UNDIME -	Releases previously allocated arrays.
SETCOR -	Define core initialization pattern.

Location: HSRI:LIBRARY

Source Language: Assembler

Return Codes: &RC4 - Error return for all entries

Subroutines Used: FREESPAC, GETSPACE, SERCOM

Logical I/O Units:

SERCOM - Error messages

Description:

After being dynamically dimensioned, an array may be used in any legitimate FORTRAN context and may also be redimensioned by the DIME or REDIME entries, or undimensioned by the DIME, REDIME, or UNDIME entries. The array may be of any type (LOGICAL, INTEGER, REAL, or COMPLEX), of any length (1,2,4,8, or 16), and dimensionality in the range (1 - 7). Type and dimensionality may not be changed in the course of the program.

An array that is to be dynamically dimensioned and allocated must satisfy the following conditions:

1) It must appear as a dummy argument in a SUBROUTINE, ENTRY, or FUNCTION statement in the highest-level routine in which it is used.

2) It must be object-time dimensioned. That is, it must appear in an explicit type or DIMENSION statement with INTEGER*4 variable dimensions. These dimension variables may be in blank or named common, or may be passed as arguments through a call on the routine.

A SUBROUTINE, ENTRY, or FUNCTION statement that references arrays to be dynamically dimensioned must satisfy the following conditions:

1) The only arguments it may reference are arrays which satisfy the previous two conditions. That is, the arguments must be capable of being dynamically dimensioned.

2) It must be in the highest level routine that will use the arrays referenced by its argument list.

3) In general, it should not be referenced by any CALL statements.

Restrictions:

1) The number of entries in use at any given time must be less than 256.

2) The number of arrays referenced by an entry must be less than 256.

3) The total amount of space requested on a single call cannot be more than 256 pages (1,048,576 bytes).

Examples:

1) Dimensioning a MAIN program

Due to FORTRAN's handling of array dimensioning, arrays to be dynamically dimensioned must be in subprograms. This in no way restricts the use of this routine, however, since a subprogram can be run in MTS as a main program.

```
SUBROUTINE MAIN(A,B,C)
REAL*8       C(N,N)
INTEGER*4    B(M,M),IDIM(7)
LOGICAL*1    A(L)
COMMON       N,M,L
 ...
Determination of N,M by input or computation
 ...
CALL DIME(0,0,4*M*M,8*N*N)
 ...
Determination of L
 ...
CALL DIME(0,L,-1,0)
 ...
Get new value for M and assign proper values to
IDIM(1) ... IDIM(7)
 ...
CALL REDIME(0,B,IDIM)
 ...
```

```
      CALL UNDIME(0)
      RETURN
      END
```

2) <u>Dimensioning for a deeper-level routine</u>

<u>Routine at higher level</u>

```
      COMMON N,M,L
        ...
      EXTERNAL ENT
        ...
      Determine N,M,L
        ...
      CALL DIME(ENT,4*N*M,4*N*M*L)
      CALL DOODAH(HONK,TWEET,SNORT)
        ...
      CALL UNDIME(ENT)
      STOP
      END
```

<u>Routine at deeper level</u>

```
      SUBROUTINE DOODAH(X,Y,Z)
        ...
      ENTRY ENT(Q,R)
      REAL*4 Q(N,M),R(N,M,L)
      COMMON N,M,L
        ...
      RETURN
      END
```

Entry Point: DIME

Module Name: DIME

Purpose:

To dynamically dimension storage and allocate new arrays.

Calling Sequence:

```
CALL DIME(ENTRY,LEN1,  ...  ,LENn,&RC4)
```

Parameters:

ENTRY	The INTEGER*4 entry point address, or "0" for the entry point of the calling program. If ENTRY references N arrays, there must be exactly N lengths passed to DIME.
LENi	The INTEGER*4 desired length in bytes of the ith array referenced by the entry.

LENi > 0 -	The space currently allocated to the ith array is released and the new amount of space, rounded upward to a multiple of 8, is obtained.
LENi = 0 -	Space currently allocated is released.
LENi < 0 -	No change is made for the ith array.

Description:

The total amount of space requested is calculated and obtained in a single block. The block is then divided among the arrays according to the lengths given. If the entry is one that has not previously been used, space for an element of the entry list is also allocated. The space to be released is released. A parameter list is constructed from the new addresses and passed to the prologue code of the subject subroutine entry. Return is made to the calling program at the statement following the call or, if an error has occurred, to the indicated label indicated by &RC4. In the event of an error return, no space has been allocated. DIME may be called any number of times for an entry.

Entry Point: REDIME

Module Name: DIME

Purpose:

To dynamically alter the dimensions of DIMEd arrays and to save all or part of the contents of the upper left corner of the altered array.

Calling Sequence:

```
CALL REDIME(ENTRY,ARRAY1,DIMS1,  ...  ,ARRAYn,DIMSn,&RC4)
```

Parameters:

ENTRY	The INTEGER*4 entry used in a previous call to DIME.
ARRAYi	The array referenced by ENTRY that is to REDIMEd.
DIMSi	An INTEGER*4 array of information: DIMSi(1) = LENi , the new length of ARRAYi in bytes. DIMSi(2) = Type of ARRAY1 (1,2,4,8, or 16). DIMSi(3) = M, the number of dimensions for ARRAYi. DIMSi(4) ... DIMSi(3+M) The old dimensions. DIMSi(4+M) ... DIMSi(3+2+M) The new dimensions.

Description:

If LENi > 0 the new space is obtained and as much of ARRAYi as will fit into the new space is saved in the upper left-hand corner. The old space is released.

If LENi = 0 the old space is released.

If LENi < 0 no change is made to ARRAYi.

ARRAYi need not be referenced in the same order as in the original call to DIME and only those being REDIMEd need to be present at all. If LENi < 0 or LENi = 0 then the array elements DIMSi(2) ... DIMSi(3+2+M) are not required for the subroutine call.

Entry Point: UNDIME

Module Name: DIME

Purpose:

To release storage previously acquired by DIME or REDIME.

Calling Sequence:

```
CALL UNDIME(ENTRY,ARRAY1,  ...  ,ARRAYn,&RC4)
```

Parameters:

ENTRY	The entry used in a previous call to DIME.
ARRAYi	An array referenced by the entry.

Description:

The space allocated to the arrays is released. Only space allocated to the arrays specifically given in the call are released, unless no arrays are specified in which case all space for all arrays referenced by the given entry is released, including the space for the entry list element. This action constitutes taking the entry out of use. Thus the call

```
     CALL UNDIME(ENTRY)
```

will release all arrays for the entry.

Entry Point: SETCOR

Module Name: DIME

Purpose:

To define an initialization pattern for space obtained by DIME or REDIME.

Calling Sequence:

```
CALL SETCOR(LEN,STRING)
```

Parameters:

LEN	The INTEGER*4 length of the pattern in STRING where LEN has values of 1 - 256.
STRING	The array containing the initialization pattern.

Description:

The core constant is changed from its previous value to the value contained in STRING. On subsequent call to DIME or REDIME this new bit pattern is used to initialize the new space. Calling SETCOR with LEN < 0 or LEN > 256 will cause the default pattern (X'81') to be restored. Thus the call

```
     CALL SETCOR(0)
```

will reset the core constant.

<u>Entry Point:</u>	DISTIM
<u>Module Name:</u>	DISTIM

<u>Purpose:</u>

To interrupt a program after an specified CPU interval, display run costs, and then restart the program.

<u>Location:</u>	HSRI:LIBRARY
<u>Source Language:</u>	370 Assembler
<u>Calling Sequence:</u>	CALL DISTIM(INTRVL)

<u>Parameters:</u>

INTRVL	The INTEGER*4 CPU time interval in seconds.

<u>Subroutines Used:</u>

COST, FWRT, SERCOM, SETIME, TIME, TIMNTRP

<u>Logical I/O Units:</u>

SERCOM - Print time and cost

<u>Restriction(s):</u>

DISTIM should only be called in TERMINAL mode.

<u>Description:</u>

The MTS timer interrupt routines are utilized to set up an interrupt every INTRVL seconds of CPU time expended. After saving all general and floating point registers, TIME and COST are called to print the current time and cost from the beginning of the current signon. The program is then started at the point of the interrupt.

<u>Example(s):</u>

```
     CALL GUINFO(10,MODE)
     IF(MODE.EQ.0) CALL DISTIM(10)
```

Entry Point: EKOLIN

Module Name: EKOLIN

Purpose:

To print a string on a specified I/O unit with folding at a specified print width and with an optional prefix.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL EKOLIN(STRING,LENGTH,BUFFER,WIDTH,PREFIX,PFXLEN,
            BRKCHR,BRKNUM,UNIT,&RC4)
```

Parameters:

STRING	The LOGICAL*1 character string to be printed.
LENGTH	The INTEGER*4 length of the information in STRING.
BUFFER	A LOGICAL*1 print buffer used by EKOLIN. Must be dimensioned at least WIDTH+1
WIDTH	The INTEGER*4 output print width.
PREFIX	A LOGICAL*1 string containing the desired prefix.
PFXLEN	The INTEGER*4 length of the prefix string in PREFIX.
BRKCHR	A LOGICAL*1 string containing a set of break characters.
BRKNUM	The INTEGER*4 number of break characters in BRKNUM.
UNIT	The logical I/O unit on which the string will be written. If UNIT = -1 or if (UNIT < -1 or UNIT > 19), then the output is written on SPRINT.

Return Code(s):

&RC4 - Output error from subroutine EWRITE

Subroutines Used:

EQUC, EWRITE, MOVEC, SETC, SPRINT

Logical I/O Units:

SPRINT - Subroutine output for UNIT = -1

Description:

The character string in the array STRING is printed out in segments of length WIDTH on the specified I/O unit. If no break characters are specified, then the string is broken at the proper width and printed in segments. If a set of break characters is specified, then the last 20 characters of the segment to be output are searched for one of the break characters. If one is found, the string is broken at this point. If a prefix string is specified, then this string is printed at the beginning of the first segment and a string of blanks of equivalent length is printed on all succeeding lines.

Example(s):

If STRING contains the 71 character string:

'The purpose of the EKOLIN subroutine is to provide a formatted printout'

then the subroutine call:

```
CALL EKOLIN(STRING,71,BUFFER,40,'String = ',9,' ',1,10)
```

will produce the following printout. The ruler is shown for convenience in this example and is not a part of the EKOLIN output.

```
         1         2         3         4
1234567890123456789012345678901234567890

 String = The purpose of the EKOLIN
          subroutine is to provide a
          formatted printout
```

Entry Point: ESCAPE

Module Name: ESCAPE

Purpose:

To intercept attention interrupts in FORTRAN programs and return control to the calling program after an interrupt.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence: CALL ESCAPE(ATTNSW,&RC4,&RC8, ...)

Parameters:

ATTNSW An INTEGER*4 variable that determines the return code of the subroutine in the case of an attention interrupt. See the description below for the behavior of the routine for ATTNSW < 0. For ATTNSW equal to or greater than zero:

```
    ATTNSW = 0 -  RC = 0
    ATTNSW = 1 -  RC = 4
    ATTNSW = 2 -  RC = 8
    etc.
```

Return Code(s):

&RCn - Return after an interrupt when ATTNSW = n/4.

Subroutines Used:

ATTNTRP, COST, FWRT, CMDNOE, GUSRIN, MTS, SERCOM, SETLIO

Logical I/O Units:

SERCOM - Notification of interrupt, cost of run and request to continue.

Description:

The MTS subroutine ATTNTRP is used to set up a trap for a single attention interrupt. When an attention interrupt occurs, MTS passes control to ESCAPE. ATTNTRP is reset for another interrupt, and SCARDS, GUSER, and *SOURCE* are all reset to *MSOURCE*.

If ATTNSW > 0 or ATTNSW = 0, a return is made as specified by the value of ATTNSW.

If ATTNSW < 0, the cost of the current run from the time of signon is computed and the message

```
XX.XX Dollars used.  Continue? (Y/N/MTS)
```

is printed on SERCOM. If "Y" is entered, the program is restarted at the point of the interrupt. If "N" is entered, a return is made as specified by the value of -ATTNSW. If "M" is entered, the subroutine MTS is called to provide a restartable return to the MTS command mode. If no commands that unload the program are issued in MTS, the command $RESTART will restart the program at the point of the interrupt.

Example(s):

```
ATTNSW = 0
CALL ESCAPE(ATTNSW,&200,&300,&400)
ATTNSW = 3
```

Note: When using the FORTRAN H-Compiler with optimization, the calling sequence shown above results in an error since it is not a logically possible sequence of events. The error can be circumvented by putting ATTNSW in COMMON to trick the compiler.

Entry Point: EWRITE

Module Name: EWRITE

Purpose:

To write an output record on a specified logical I/O unit using the MTS WRITE subroutine with the facility for trapping I/O and file assignment errors.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

CALL EWRITE(REGION,LEN,MOD,LNUM,UNIT,&RC4)

Parameters:

Definitions of the parameters are the same as those of the MTS WRITE subroutine. See the MTS Manual, Volume 3 for more information.

REGION	The location of the region from which data will be transmitted.
LEN	The INTEGER*2 length (in bytes) of the data in REGION.
MOD	The INTEGER*4 modifier used to control the action of the subroutine.
LNUM	An INTEGER*4 variable giving the internal value of the line number that is to be written, or that has been written.
UNIT	The INTEGER*4 FDUB pointer or logical I/O unit number, or a left-justified 8-character logical I/O unit name (i.e., 'SPUNCH ').

Return Code(s):

&RC4	File assignment error for the specified UNIT, or non-zero return code from WRITE.

Subroutines Used: FREESPAC, GDINFO, SERCOM, WRITE

Logical I/O Units:

SERCOM	Error messages
WRITE	Data output

Description:

The modifier supplied with the subroutine call is OR'ed with the NOPROMPT and ERRRTN modifiers and WRITE is called with this altered modifier. If a file assignment error is encountered, a message is printed on SERCOM and RC4 is taken. If a non-zero return code from the WRITE operation occurs, GDINFO is called to retrieve the associated I/O error message. If the message can be located, it is printed on SERCOM and RC4 is taken.

Entry Point: FAIL

Module Name: FAIL

Purpose:

To intercept program interrupts in FORTRAN programs and to return control to the calling program.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence: CALL FAIL(PPRNT,&RC4)

Parameters:

PPRNT	A INTEGER*4 I/O unit number (1 - 19) where a loader map will be written in case of an interrupt, or the value "0" for no dump.

Return Code(s):

&RC4 - The statement number to branch to in case of an interrupt.

Subroutines Used:

CMDNOE, LODMAP, PGNTTRP, SERCOM, WRITE

Logical I/O Units:

SERCOM -	Notification of interrupt and PSW.
WRITE -	Notification of interrupt, PSW, and loader map.

Description:

On the initial call to FAIL, the MTS routine PGNTTRP is called to set up the interrupt trap and the routine returns normally. If a program interrupt occurs at a later time, MTS return control to FAIL. The PSW is decoded and put into standard form and a message of the form

```
+PROGRAM INTERRUPT+
PSW =   071D0005 A081CD8E
```

is printed on SERCOM. If PPRNT = 0 a return is made to the program via RC4.

Description: (Continued)

If PPRNT > 0, the SERCOM error message is printed on PPRNT along with the loader map at the time of the interrupt. Then the $MESSAGE system is used to send a notification message of the interrupt to 'UMTRI' and a return to the calling program is made via &RC4.

Entry Point: FILLM

Module Name: FILLM

Purpose:

To perform multiple fill operations with a single subroutine call.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
CALL FILLM(ARRAY1,LEN1,CHAR1,  ...  ,ARRAYn,LENn,CHARn)
```

Parameters:

ARRAYi	The array location where filling will begin.
LENi	The INTEGER*4 number of bytes to fill.
CHARi	The LOGICAL*1 character used to fill the array.

Description:

With the exception of a different calling sequence, this routine is similar to the MTS routine SETC except that multiple fill operations are possible with a single subroutine call.

Entry Point:	FIVPAR
Module Name:	FIVPAR

Purpose:

To convert an OSIRIS type 5 dictionary record into binary variables.

Location:	HSRI:LIBRARY
Source Language:	FORTRAN

Calling Sequence:

```
CALL FIVPAR(DICREC,VARNUM,VARNAM,CHRTYP,RECLOC,FLDWTH,
            NUMDEC,NUMRES,MISONE,MISTWO)
```

Parameters:

DICREC	The LOGICAL*1 80-character type 5 dictionary record.
VARNUM	The INTEGER*2 variable number.
VARNAM	The LOGICAL*1 24-character variable name.
CHRTYP	The INTEGER*2 character type 0 - character numeric 1 - alphabetic 2 - fullword integer binary 3 - fullword floating point binary 4 - packed decimal 5 - zoned decimal 6 - halfword integer binary
RECLOC	The INTEGER*2 record location
FLDWTH	The INTEGER*2 field width
NUMDEC	The INTEGER*2 number of implied decimal places
NUMRES	The INTEGER*2 number of responses
MISONE	The INTEGER*4 missing data code #1
MISTWO	The INTEGER*4 missing data code #2

Subroutines Used:

DTB, EQUC, IGC, MOVEC

Description:

The character values are converted into their binary representations. If the missing data code fields are blank on the input record, then the value 1,500,000,000 is returned for MISONE and/or MISTWO.

Entry Point:	FLOT
Module Name:	FLOT

Purpose:

To convert a fixed binary number with an implied number of decimal places to floating point representation.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler
Calling Sequence:	CALL FLOT

Parameters:

GR0	The number to be converted
GR1	The number of decimal places in the resultant floating point value.
FR1	The converted floating point number.

Restriction(s):

This routine may only be called from an assembler program.

Description:

FLOT converts fixed binary to floating point. It will convert character representations of numbers up to 16,777,215 accurately. Numbers that are larger than this will lose precision in the low order digits.

Entry Point:	FWRT
Module Name:	FWRT

Purpose:

To convert REAL*4 numbers to a character format.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL FWRT(ARRAY,START,LENGTH,NUMDEC,NUMBER,&RC4)
```

Parameters:

ARRAY	The LOGICAL*1 array where the character number will be placed.
START	The INTEGER*4 location in ARRAY where the number begins.
LENGTH	The INTEGER*4 length of the character number. Length must be greater than 1 and less than 17.
NUMDEC	The INTEGER*4 number of decimal places in the output.
NUMBER	The REAL*4 binary number to be converted.

Return Code(s):

&RC4 - The character number is too large for the output field width specified.

Description:

This routine converts signed REAL*4 binary numbers to decimal characters with an imbedded decimal point. The character number is right justified in the output array with leading blanks. If the decimal representation of the number is too long for the specified field width, the output array is filled with asterisks.

<u>Entry Point:</u> GETDAT

<u>Module Name:</u> GETDAT

<u>Purpose:</u>

To set up the input data set by calling INFILE and ININ

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

CALL GETDAT(LRECL,&RC4)

<u>Parameters:</u>

LRECL	The INTEGER*4 record length of the input data set as returned from ININ.

<u>Return Code(s):</u>

&RC4 - Error in converting the file number, record length, or blocking factor, or an error return from INFILE, ININ, or CLOIN

<u>Subroutines Used:</u>

DTB, IGC, INFILE, MOVEC, SETC

Entry Point: GETTP

Module Name: GETTP

Purpose:

To mount tapes or access files while a program is running. Access information is read by the routine during execution.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL GETTP(UNIT,PAR1,PAR2,&RC4)

Parameters:

UNIT	An INTEGER*4 array of dimension 1 - 6 containing the FDUB for the ith device acquired.
PAR1	The INTEGER*4 number of devices to be obtained on a particular call. PAR1 must be in the range 0 - 6. If PAR1 = 0, then device control cards will be read but no new devices will be obtained.
PAR2	An INTEGER*4 switch specifying whether to look for control cards or not. PAR2 = 0 - Look for control cards. PAR2 = 1 - Don't look for control cards.

Return Code(s):

&RC4 - Error in MOUNT or CONTROL subroutines.

Subroutines Used:

ADROF, CNTRL, DTB, GETFD, GUINFO, GUSER, ICLC, ITRT, MOUNT, MOVEC, RCALL, SCARDS, SERCOM, SETC, SPRINT

Logical I/O Units:

GUSER -	Read control card input after an error.
SCARDS -	Read control card input.
SERCOM -	Print program prompts after an error.
SPRINT -	Print program prompts.

Description:

GETTP will permit the mounting of tapes or the acquisition of other files while a program is running, and optionally will allow the user to supply tape control commands to tapes that have been mounted. GETTP reads control cards from SCARDS. The control cards have the following format:

Col 1: Device number (1 - 6)
Col 2 - 80: Mount request, file name, or device control command.

If the control card contains a mount request, the tape will be mounted and the FDUB for the pseudo-device name specified will be placed in the corresponding location in UNIT. If the control card contains a file name, an FDUB will be acquired and placed in UNIT. After GETTP has obtained all the devices requested, it will optionally read control cards from SCARDS until an end-of-file is encountered, executing each control command via the MTS CONTROL subroutine.

Example(s):

```
      SUBROUTINE MYPROG
      INTEGER*4   UNIT(3)
       ...
      CALL GETTP(UNIT,3,0,&999)
       ...
      RETURN
       ...
  999 RETURN 1
      END
```

To execute the program the following setup may be used as a $SOURCE or batch file.

```
$RUN MYPROG SCARDS=*SOURCE*
1C1234A 9TP *S1* VOL=TAPE01 'FIRST'
2C2345B 9TP *S2* WRITE=YES VOL=TAPE02 'SECOND'
3-TEMP
1POSN *5*
2POSN *EOT*
2DSN NEWFILE
 ...
$ENDFILE
```

Entry Point: GUSRIN

Module Name: GUSRIN

Purpose:

To read input into an array on I/O unit GUSER with the features of line continuation, upper-case conversion, and array length protection.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL GUSRIN(STRING,LENGTH,MAXLEN,&RC4,&RC8,)
```

Parameters:

STRING	A LOGICAL*1 array of dimension MAXLEN.
LENGTH	A INTEGER*4 variable that contains, on exit from the routine, the length of the input string plus one for the trailing blank added to the end of the string.
MAXLEN	The INTEGER*4 length of STRING.

Return Code(s):

&RC4	An end-of-file was encountered.
&RC8	The input line length is greater than MAXLEN-1

Subroutines Used:

ADROF, BTD, EQUC, GDINF, GUSER, LAND, LOR, MOVEC, RCALL, SERCOM, SETLIO, SETPFX

Logical I/O Units:

GUSER	Read input string
SERCOM	Error messages

Description:

This subroutine is intended for user error replacement input. The prefix character for the read is set to "?". The read is made on GUSER with the modifiers @TRIM, @CASECONV, @MAXLEN, and @NOTIFY. If the input string ends in the continuation character "-", another read is made and the new characters are added on to the end of the previous string beginning at the location of the "-". If the total number of characters read in is greater than MAXLEN-1, a branch is made to RC8. If not, the prefix character is reset, a trailing blank is appended to the string, and a normal return is taken.

If an I/O unit re-assignment occurs, the GDINF subroutine is called to determine what the assignment is. If the assignment is not to *MSOURCE*, then an error message is printed and GUSER is assigned to *MSOURCE*. The read is then performed as described above.

Entry Point: GUSRNC

Module Name: GUSRIN

Purpose:

To read input into an array on I/O unit GUSER with the features of upper-case conversion and array length protection. The action is the same as the GUSRIN entry with the exception that line continuation is not supported.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL GUSRNC(STRING,LENGTH,MAXLEN,&RC4,&RC8)

Parameters:

STRING	A LOGICAL*1 array of dimension MAXLEN.
LENGTH	A INTEGER*4 variable that contains, on exit from the routine, the length of the input string plus one for the trailing blank added to the end of the string if no continuation character is present. If there is a continue character at the end of the segment, then LENGTH contains the length of the input string including the continue character.
MAXLEN	The INTEGER*4 length of STRING.

Return Code(s):

&RC4	An end-of-file was encountered.
&RC8	The input line length is greater than MAXLEN-1

Subroutines Used:

ADROF, BTD, EQUC, GDINF, GUSER, LAND, LOR, MOVEC, RCALL, SERCOM, SETLIO, SETPFX

Logical I/O Units:

GUSER	Read input string
SERCOM	Error messages

Description:

This entry performs the same function as GUSRIN except for line continuation. A trailing dash "-" is ignored by the routine and treated as any other character. If a continue character is found at the end of the line, no trailing blank is appended to the line.

This routine is intended for use in dynamic dimensioning applications where an very long string is read in segments and a buffer array of unknown length must be generated dynamically to hold the information. In such a case, the line continuation must be performed outside of GUSRIN as part of the dimensioning process.

Because this routine is intended as a user error correction input, GUSER is always attached to *MSOURCE* and cannot be reassigned.

Entry Point: IFILTR

Module Name: IFILTR

Purpose:

To read user supplied FILTER, RECODE, and TITLE statements, to check the statement syntax, decode the FILTER and RECODE statements, and to call the APIN entries SETFIL and SETREC.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL IFILTR(TITLE,TITLEN,&RC4)

Parameters:

TITLE	The LOGICAL*1 132 character title entered by the user.
TITLEN	The INTEGER*4 length of the title in TITLE.

Return Code(s):

&RC4 - Error in processing the filter, recode or title.

Subroutines Used:

BTD, DIME, DTB, EKOLIN, EQUC, EWRITE, FINDC, FIVPAR, FRDICT, GUSRIN, IFRER, IGC, LCOMC, LYSOMB, MOVEC, PRNTCK, READ, READIN, SERCOM, SETC, SPRINT

Logical I/O Units:

READ -	Read the on-line dictionary
SERCOM -	Error messages
SPRINT -	Program input requests

Description:

Requests for filter, recode, and title statements are made on SPRINT. The input is read into a dynamically dimensioned array by IFRER and decoded by INTER or RECSYN. Lists of the filter or recode variables requested are printed by FRDICT. The entries SETFIL and SETREC are called to pass the decoded parameters to APIN for subsequent use in the data entry operations.

<u>Module Name:</u> ILABEL

<u>Purpose:</u>

This set of routines provides for the retrieval of code value labels from a special label file created by the program HSRI:LABGEN. There are four entry points:

ILABEL -	To initialize the routine and open the label file.
CLABEL -	To check if labels exist for a particular variable.
GLABEL -	To retrieve code value labels
GVAR -	To retrieve variable names.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Subroutines Used:</u> FREEFD, FREESPAC, GDINFO, GETFD, READ

<u>Description:</u>

This module permits programs to retrieve variable names and code value labels from a specially prepared label file for on-line documentation purposes. The label file must be constructed by the HSRI:LABGEN program either by direct entry of the code value label information, or by use of the LABEL command in the HSRI:CODEBOOK program.

<u>Entry Point:</u> ILABEL

<u>Module Name:</u> ILABEL

<u>Purpose:</u>

To check the label file and open it for subsequent use by CLABEL, GLABEL, and GVAR.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Calling Sequence:</u> CALL ILABEL(FDNAME,LABSW,&RC4)

<u>Parameters:</u>

FDNAME	A LOGICAL*1 array containing the name of the label file. The name must be terminated by a trailing blank.
LABSW	A LOGICAL*1 switch giving the status of the label file. (see description)

<u>Return Code(s):</u>

&RC4 - Bad label file name.

<u>Description:</u>

If the file name is blank, a normal return is made with LABSW = .FALSE.

If the name is not blank, an FDUB is acquired, and GDINFO is called to see if the file exists and is a line file. A further check of line 0 of the file is made to see if the file is a label file. If any trouble is encountered in these checks, an RC4 return is taken with LABSW = .FALSE.

If the label file specified is valid, a normal return is taken with LABSW = .TRUE.

Entry Point:	CLABEL
Module Name:	ILABEL

Purpose:

To check if labels exist for a particular variable number.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL CLABEL(VARNUM,LENGTH,MAXCOD,&RC4)
```

Parameters:

VARNUM	The INTEGER*4 variable number.
LENGTH	The INTEGER*4 maximum length of the code value labels for VARNUM.
MAXCOD	The INTEGER*4 maximum code value for VARNUM.

Return Code(s):

&RC4 - No code value labels for VARNUM.

Description:

The entry CLABEL checks to see if code value labels exist for variable VARNUM. If they do, the maximum code value and the maximum code value length are returned. If there are no code labels for VARNUM, an RC4 is taken.

Entry Point: GLABEL

Module Name: ILABEL

Purpose:

To retrieve the code value label for a specified variable number and code value from the label file.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

CALL GLABEL(VARNUM,CODNUM,LABEL,JUST,&RC4,&RC8,&RC12)

Parameters:

VARNUM	The INTEGER*4 variable number.
CODNUM	The INTEGER*4 code value for VARNUM.
LABEL	A LOGICAL*1 array of at least 16 bytes where the code value will be placed.
JUST	A INTEGER*4 value defining label justification: 0 - Label is left-justified. 1 - Label is right-justified.

Return Code(s):

&RC4 -	No code value labels for VARNUM
&RC8 -	CODNUM greater that maximum code value.
&RC12 -	No code value label for CODNUM.

Description:

GLABEL finds the code value label for CODNUM for variable VARNUM and returns it in the first LENGTH bytes (LENGTH is defined by CLABEL) of the array LABEL. The label is left or right justified within the LENGTH bytes as specified by the variable JUST.

Example(s):

If the value of LENGTH for a variable is 6, then GLABEL will place the following in a 16-byte array LABEL for the code value label 'YES':

```
     JUST = 0     'YES   aaaaaaaaaa'
     JUST = 1     '   YESaaaaaaaaaa'
```

where the bytes denoted by 'a' are not accessed by GLABEL.

Entry Point:	GVAR
Module Name:	ILABEL

Purpose:

To return the 24-character dictionary name for a specified variable number.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL GVAR(VARNUM,LENGTH,MAXCOD,VARNAM,&RC4,&RC8,&RC12)
```

Parameters:

VARNUM	The INTEGER*4 variable number.
LENGTH	The INTEGER*4 code value label length for VARNUM.
MAXCOD	The INTEGER*4 maximum code value for VARNUM.
VARNAM	A LOGICAL*1 24-byte array where the variable name will be placed.

Return Code(s):

&RC4 -	No code labels for this variable
&RC8 -	Variable not found.
&RC12 -	VARNUM > maximum variable number.

Description:

The variable name for the specified variable number is read from the label file and placed in the VARNAM array. The variables LENGTH and MAXCOD are not used in this entry.

Module Name: INFILE

Purpose:

To provide an interface between programs that access files and the actual read, write, and control routines. The subroutine supports fixed-block type data records and performs most of the actual file support operations through the QSAM routines.

There are five entry points to the module:

INFILE -	To acquire up to four files for subsequent input operations.
ININ -	To open a given file for input.
GETCHA -	To read the next sequential record from the file.
CLOIN -	To close the given file.
FREEIN -	To release the file. Subsequent input operations will require another call to INFILE.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Subroutines Used:

BTD, CHKFIL, DTB, EQUC, IGC, LCOMC, MOVEC, QSAM, SERCOM, SETC

Logical I/O Units:

SERCOM - Error messages

Description:

These routines will read sequential fixed-block records from a tape or disk file. If the file is on disc, it may be labelled (as generated by OUTFIL) or unlabelled. If the file is on tape, all control operations necessary to position the tape to the desired DSN and to obtain the necessary blocking information are handled by INFILE.

Entry Point: INFILE

Module Name: INFILE

Purpose:

To set up input files for later use by ININ, GETCHA, and CLOIN.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL INFILE(DSR,PDNAME,VOLUME,DSNAME,FILENO,LRECL,BLKFAC,
           &RC4)
```

Parameters:

DSR	An INTEGER*4 file reference number (1-4). This number serves as an index to the input unit for all subsequent operations.
PDNAME	The LOGICAL*1 pseudo-device name if the file resides on tape, or blank if the file resides on disk. If not blank, the name must be 3 to 16 characters in length and must terminate with a trailing blank.
VOLUME	The LOGICAL*1 6 character volume serial name for the pseudo-device specified by PDNAME, or blank for disc files.
DSNAME	The LOGICAL*1 DSN if the file resides on tape, or the file name if the file resides on disk. The name must be 1 - 17 characters in length and must terminate with a trailing blank.
FILENO	The INTEGER*4 file designator. For tapes = the file number, or "0". For disc = "0" for labelled files = "1" for unlabelled files.
LRECL	The INTEGER*4 logical record length for unlabelled files, or "0" for labelled files.
BLKFAC	The INTEGER*4 blocking factor for unlabelled files, or "0" for labelled files.

Return Code(s):

&RC4 - A wide variety of errors that occur in setting up the file. Error returns are generally preceded by a message from INFILE or from QSAM.

Description:

If a disk file is specified (as indicated by a blank PDNAME), the file name is checked for validity, an FDUB is acquired, and the file parameters are stored in internal arrays.

If a previously used tape is specified, an FDUB is acquired, the volume name of the tape is checked against the name supplied, the current tape position is determined, and the file parameters are stored in internal arrays.

If a previously unused tape is specified, the same operations described above are performed, but in addition blocking is disabled on the tape (i.e., BLK=OFF).

Entry Point: ININ

Module Name: INFILE

Purpose:

To open a file acquired by INFILE for subsequent read operations.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL ININ(DSR,LRECL,&RC4)
```

Parameters:

DSR	The INTEGER*4 file reference number used in a previous call to INFILE.
LRECL	The INTEGER*4 logical record length. On input the logical record length for unlabelled disk files if not supplied in the INFILE entry. On output the logical record length for labelled disc files or tapes as determined from the file header record.

Return Code(s):

&RC4 - Many errors resulting from tape positioning, invalid header records, etc.

Description:

If the file is an unlabelled disk file, the file is opened for reading with the specified logical record length.

If the file is a labelled disk file, The header record(s) is read to determine its validity and the record length and blocking factor are acquired. The file is then opened for reading.

If the file is on tape, the tape is positioned to the specified file number if this value in non-zero, or to the specified DSN if the file number is zero. The blocking factor and record length are determined and the file is opened for reading.

<u>Entry Point:</u> GETCHA

<u>Module Name:</u> INFILE

<u>Purpose:</u>

To read the next sequential record from the input file.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

```
CALL GETCHA(DSR,DATA,&RC4,&RC8)
```

<u>Parameters:</u>

DSR	The INTEGER*4 file reference number used in a previous call to INFILE.
DATA	The region where the data record from the input file will be placed.

<u>Return Code(s):</u>

&RC4 - End-of-file from the input unit.

&RC4 - Error return from QGET

<u>Description:</u>

The subroutine entry QGET is called to get the next record from the input file.

<u>Entry Point:</u> CLOIN

<u>Module Name:</u> INFILE

<u>Purpose:</u>

To close the specified input file.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

```
CALL CLOIN(DSR,&RC4)
```

<u>Parameters:</u>

DSR	The file reference number used in a previous call to INFILE.

<u>Return Code(s):</u>

&RC4 - Control error while positioning tape.

<u>Description:</u>

If the file is on disc, it is closed. If the file is on tape, the tape is positioned to the file number defined in the last call to ININ and the file is closed.

Entry Point: FREEIN

Module Name: INFILE

Purpose:

To release the file acquired by entry INFILE.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL FREEIN(DSR,&RC4)

Parameters:

DSR	The file reference number used in a previous call to INFILE.

Return Codes:

&RC4 - Invalid DSR number.

Description:

If the file is on disc, a check is made to see if another DSR uses the same DSN. If not, then the FDUB is released. All internal arrays for this DSR are initialized.

If the file is on tape, a check is made to see if another DSR uses the same tape. If not, the FDUB is released and the internal arrays are initialized.

Entry Point:	INFOF
Module Name:	INFOF

Purpose:

To check the type of file assigned to an I/O unit.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler
Calling Sequence:	ITYPE = INFOF(UNIT,TYPE)

Parameters:

UNIT	The I/O unit specification. For TYPE = 0, UNIT is the INTEGER*4 logical I/O unit number (1 - 19) or an FDUB. If TYPE = 1 then UNIT is an eight character I/O unit name with trailing blanks (i.e., 'SPUNCH ')
TYPE	The INTEGER*4 type of unit specification.
ITYPE	The INTEGER*4 file type assigned to UNIT.

```
    ITYPE = 1 - Line file
          = 2 - Sequential file
          = 3 - Not a line or sequential file
          = 4 - UNIT not assigned or bad FDUB.
```

Subroutines Used: FREESPAC, GDINFO

Description:

This subroutine calls GDINFO and compares word 2 with the types 'FILE' and 'SEQF'. If these are found INFOF is assigned the value "1" or "2" respectively. For all other cases INFOF is assigned the value "3".

Example:

```
    IF(INFOF(10,0).NE.1) RETURN 1
```

Entry Point: ITRNSL

Module Name: ITRNSL

Purpose:

To translate one character string into another.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
IRETRN = ITRNSL(ARRAY,NTRANS,LENOLD,LENNEW,OLD,NEW)
```

Parameters:

ARRAY	A LOGICAL*1 array containing the translation strings. Each old character string is followed by the new character string for that case.
NTRANS	The INTEGER*4 number of translations possible. That is, the number of old string/new string pairs in ARRAY.
LENOLD	The INTEGER*4 length of the old string to be translated. (LENOLD.LE.256)
LENNEW	The INTEGER*4 length of the new translated string. (LENNEW.LE.256)
OLD	The LOGICAL*1 array containing the string to be translated.
NEW	The LOGICAL*1 array where the translated string will be placed. NEW remains unchanged if the string in OLD does not occur in ARRAY.
IRETRN	An INTEGER*4 variable specifying the results of the translation. A value of zero indicates that the translation was successful.

Description:

This subroutine is similar in action to the MTS routine TRNC except that strings are translated to strings and the input and output strings need not be of equal length.

<u>Example(s):</u>

```
     ITRNSL('YES1NO 2   9',3,3,1,ANSWER,OPT)

     ITRNSL('FORD01CHEV02BUIC03PLYM04    99',5,4,2,IN,OUT)
```

Entry Point:	IWRT
Module Name:	IWRT

Purpose:

To perform a binary to character number conversion with optional prefix and suffix characters.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL IWRT(ARRAY,START,LENGTH,NUMBER,1,&RC4)

CALL IWRT(ARRAY,START,LENGTH,NUMBER,2,PCHAR,&RC4)

CALL IWRT(ARRAY,START,LENGTH,NUMBER,3,SCHAR,&RC4)

CALL IWRT(ARRAY,START,LENGTH,NUMBER,4,PCHAR,SCHAR,&RC4)
```

Parameters:

ARRAY	The LOGICAL*1 array where the character number will be placed.
START	The INTEGER*4 location in ARRAY where the number should start.
LENGTH	The INTEGER*4 field width of the output number. LENGTH must be less than 17.
NUMBER	The INTEGER*4 number to be converted.
PCHAR	A prefix character to be placed before the converted number.
SCHAR	A suffix character to be placed after the converted number.

Return Code(s):

&RC4 - The number is too large for the output field.

Description:

The decimal number is placed into the output field right-justified with leading blanks. Optionally, a prefix character, a suffix character, or both may be included with the number in the output field. If the binary number is too large for the output field width specified, the output is filled with asterisks

Entry Point:	JULDAT
Module Name:	JULDAT

Purpose:

To convert Gregorian dates to the corresponding Julian date using the MTS routine GRJLDT, and to provide validation of the input data prior to conversion.

Location:	HSRI:LIBRARY
Source Language:	FORTRAN
Calling Sequence:	CALL JULDAT(IN,OUT,MODE)

Parameters:

IN	A LOGICAL*1 array of at least 6 bytes containing the Gregorian date in the form specified by MODE.
OUT	A LOGICAL*1 array of at least 5 bytes where the Julian date will be placed.

MODE	The INTEGER*4 format of the Gregorian date
	MODE = 1 IN = MMDDYY
	MODE = 2 IN = YYMMDD

Subroutines Used: BTD, DTB, GRJLDT, IGC, MOVEC

Restriction(s):

Leap years are not taken into account when checking the input dates for validity. Consequently, the entry of February 29 for a non-leap year would result in the Julian date for March 1.

Description:

The input field is first checked for non-numeric characters. Then values for year, month, and day are range-checked. In addition, the resulting Julian date is checked to see if it lies in the range of (1 - 99999). If any errors are detected, the Julian date is set to the value "99999".

Entry Point: KEYSCN

Module Name: KEYSCN

Purpose:

To scan a character string for the existence of members of a predefined set of modifiers and/or keyword phrases.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL KEYSCN(STRING,IBEG,ILAST,MAXLEN,NMOD,MOD,MPARS,MODSW,
            NKEY,KEY,KPARS,KEYSW,KEYVAL,&RC4,&RC8)
```

Parameters:

STRING	A Logical*1 array of dimension MAXLEN that contains the character string to decode. The string must terminate with a trailing blank.
IBEG	The Integer*4 location in STRING where scanning is to begin.
ILAST	The Integer*4 number of the last character in STRING.
MAXLEN	The Integer*4 dimension of STRING.
NMOD	The Integer*4 number of possible modifiers that are defined.
MOD	A Logical*1 array of dimension 8*NMOD containing the left-justified, 8-byte-aligned modifier names.
MPARS	An Integer*2 array of dimension (2,NMOD). MPARS(1,J) is the minimum number of characters that will be recognized as an abbreviation for modifier number J. MPARS(2,J) is the full length of the name for modifier number J.
MODSW	An Logical*1 array of dimension NMOD. MODSW(J) is .TRUE. if the modifier J is present in STRING and .FALSE. otherwise.
NKEY	The Integer*4 number of possible keywords that are defined.

KEY	A Logical*1 array of dimension 8*NKEY containing the left-justified, 8-byte-aligned keyword names.
KPARS	An Integer*2 array of dimension (2,NKEY). KPARS(1,J) is the minimum number of characters that will be recognized as an abbreviation for keyword number J. MPARS(2,J) is the full length of the name of keyword number J.
KEYSW	A Logical*1 array of dimension NKEY. KEYSW(J) is .TRUE. if the keyword J is present in STRING and .FALSE. otherwise.
KEYVAL	An Integer*4 array of dimension (2,NKEY). KEYVAL(1,J) gives the location in STRING where the Right-Hand-Side of keyword J begins. KEYVAL(2,J) gives the length of this RHS.

Return Code(s):

&RC4	BATCH mode syntax error.
&RC8	The replacement string was CANCELed.

Subroutines Used:

EQUC, FINDC, FINDST, GUINFO, GUSRIN, IGC, LCOMC, LSTPAR, MOVEC PRNTCK, SERCOM, SHFTST, SPLCHK

Logical I/O Units:

SERCOM Error messages

Special Note:

A 140-byte COMMON area named /BUF/ is used for I/O operations and other temporary tasks.

Description:

The array STRING is scanned for modifiers (i.e., ALLV, PRINT) and keywords (i.e., FILE=-A, VAR=1,3,5-12) that match those in the input lists defined by MOD and KEY. If a syntax error is encountered in BATCH mode, an error message is printed and the scanning of STRING continues. If a syntax error is encountered in TERMINAL mode, an attempt is made to determine if the invalid parameter is a possible misspelling of a correct one. The user is queried for verification or rejection of each possible misspelling that is found. If this process is unsuccessful, an error message is printed and the user is prompted for a replacement string, a carriage return to ignore the incorrect item, or the word CANCEL to cancel the entire scanning process. If a replacement string is supplied,, the incorrect item is shifted out of STRING and the replacement is added at the end of the modified array providing that the array dimension is not exceeded. If the replacement prompt is returned, the incorrect item is simply deleted. On exit from KEYSCN, therefore, STRING contains a corrected version of the input.

A space (X'40') is the normal break character separating modifiers and keywords.

Entry Point:	KEYREP
Module Name:	KEYSCN

Purpose:

To enter a replacement for a keyword whose RHS has been found to be invalid during the process of decoding.

Location:	HSRI:LIBRARY
Source Language:	FORTRAN

Calling Sequence:

```
CALL KEYSCN(STRING,ILAST,MAXLEN,NMOD,MODSW,NKEY,KEYSW,
            KEYVAL,IKEY,&RC4,&RC8)
```

Parameters:

STRING	A Logical*1 array of dimension MAXLEN containing the character string decoded by KEYSCN.
ILAST	The Integer*4 number of the last character in STRING.
MAXLEN	The Integer*4 dimension of STRING.
NMOD	The Integer*4 number of possible modifiers that are defined.
MODSW	An Logical*1 array of dimension NMOD. MODSW(J) is .TRUE. if the modifier J is present in STRING and .FALSE. otherwise.
NKEY	The Integer*4 number of possible keywords that are defined.
KEYSW	A Logical*1 array of dimension NKEY. KEYSW(J) is .TRUE. if the keyword J is present in STRING and .FALSE. otherwise.
KEYVAL	An Integer*4 array of dimension (2,NKEY). KEYVAL(1,J) gives the location in STRING where the Right-Hand-Side of keyword J begins. KEYVAL(2,J) gives the length of this RHS.
IKEY	An Integer*4 variable that designates the number of the keyword that requires replacement.

Return Code(s):

&RC4	BATCH mode syntax error.
&RC8	The replacement string was CANCELed.

Subroutines Used:

EQUC, FINDC, FINDST, GUINFO, GUSRIN, IGC, LCOMC, LSTPAR, MOVEC, PRNTCK, REPMSG, SERCOM, SHFTST, SPLCHK

Logical I/O Units:

SERCOM	Error messages

Special Note:

A 140-byte COMMON area named /BUF/ is used for I/O operations and other temporary tasks.

Description:

The keyword and corresponding RHS designated by IKEY is deleted from STRING and the characters in STRING are shifted left to fill the gap. If sufficient room is available, a replacement is read from GUSER and appended to the end of STRING. The new contents are then decoded as in KEYSCN in order to provide updated values of MODSW, KEYSW, and KEYVAL. On exit from KEYREP, program control should pass to the next executable statement after the KEYSCN call, just as if a normal return from that subroutine had been made.

Entry Point:	LYSOB LYSOMB
Module Name:	LEFJ

Purpose:

To left-justify a character string and remove all blanks.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

NCHAR = LYSOB(ARRAY,START,LENGTH)

NCHAR = LYSOMB(ARRAY,START,LENGTH)

Parameters:

ARRAY	The array containing the character string.
START	The INTEGER*4 location of the character in ARRAY where justification should start.
LENGTH	The INTEGER*4 number of characters in ARRAY to be checked.
NCHAR	The INTEGER*4 number of non-blank characters found.

Description:

The routine left-justifies a character string and deletes all blanks. There are two entries: the LYSOMB entry allows for primes within the character string and only deletes those blanks that are not enclosed within the primes.

Entry Point: LISFIV

Module Name: LISFIV

Purpose:

To list type 5 dictionary records on a specified I/O unit.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL LISFIV(DICREC,UNIT,&RC4)
```

Parameters:

DICREC	The LOGICAL*1 80-character type 5 dictionary record to be listed.
UNIT	The INTEGER*4 I/O unit number on which the list is to written.

Return Code(s):

&RC4 - Error return from EWRITE

Subroutines Used:

EWRITE, MOVEC, SETC

Description:

The dictionary elements are moved into a readable format and written on UNIT. Each call to LISFIV lists only one dictionary record. The entry LISHDR should be called first to produce a heading for the list.

Entry Point: LISHDR

Module Name: LISFIV

Purpose:

To print a list header on a specified I/O unit.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL LISHDR(UNIT,&RC4)

Parameters:

UNIT	The INTEGER*4 I/O unit number on which the header is to written.

Return Code(s):

&RC4 - Error return from EWRITE

<u>Entry Point:</u> LNBTD

<u>Module Name:</u> LNBTD

<u>Purpose:</u>

To convert an MTS internal file line number to character format.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

```
CALL LNBTD(LINENO,ARRAY,WIDTH)
```

<u>Parameters:</u>

LINENO	The INTEGER*4 MTS internal line number.
ARRAY	The LOGICAL*1 array where the number will be placed.
WIDTH	The INTEGER*4 width of the character string generated.

<u>Subroutines Used:</u> BTD, SETC

<u>Description:</u>

If the internal line number is a multiple of 1000 (i.e., an integral line number) the number is written with no decimal point. Otherwise, a decimal point and three decimal places are written.

<u>Example(s):</u>

Internal line number	Output
1000	1
37458000	37458
458123	458.123
6200	6.200

Entry Point: LSTFIX

Module Name: LSTFIX

Purpose:

To sort a list of numbers and delete duplicate values.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence: CALL LSTFIX(LIST,NUMBER)

Parameters:

LIST	The INTEGER*2 list of values.
NUMBER	The INTEGER*4 number of values in LIST.

Subroutines Used: CHSRT

Description:

The list of number is first sorted, then duplicate values are deleted. On output, NUMBER contains the number of non-duplicate values in LIST.

Entry Point: LSTPAR

Module Name: LSTPAR

Purpose:

To list keywords and/or modifiers for the CMDSCN and KEYSCN routines.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence: CALL LSTPAR(NWRDS,WRDPAR,WORD)

Parameters:

NWRDS	The INTEGER*4 number of words in WORD.
WRDPAR	The INTEGER*2 array of word lengths of dimension (2,NWRDS) used in CMDSCN and KEYSCN.
WORD	The LOGICAL*1 array containing the 8-character left-justified words to list.

Subroutines Used: MOVEC, SETC

Description:

This routines lists valid modifiers and/or keywords from the input arrays to CMDSCN and KEYSCN for the HELP function in error replacement.

<u>Entry Point:</u> MOVBUT

<u>Module Name:</u> MOVBUT

<u>Purpose:</u>

To move a number of strings containing numeric characters.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Calling Sequence:</u>

```
CALL MOVBUT(LEN1,IN1,OUT1,  ...  ,LENn,INn,OUTn,0)
```

<u>Parameters:</u>

LENi	The INTEGER*4 number of characters to be moved for operation "i", or "0" to terminate the sequence of moves. (LENi.LE.256)
INi	The array containing the characters to be moved.
OUTi	The array to which the characters will be moved.

<u>Description:</u>

This subroutine is similar in function to MOVEM except that only strings of numeric characters are moved. If the array INi contains any non-numeric characters, the array OUTi remains unchanged by the subroutine operation.

Entry Point:	MOVEM
Module Name:	MOVEM

Purpose:

To perform multiple move operations with a single subroutine call.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL MOVEM(LEN1,IN1,OUT1,  ...  ,LENn,INn,OUTn,0)
```

Parameters:

LENi	The INTEGER*4 number of bytes to be moved for operation "i", or "0" to terminate the move sequence.
INi	The array containing the data to be moved.
OUTi	The array to which the data will be moved.

Description:

This subroutine is identical to the MTS routine MOVEC except that it allows for multiple move operations with a single subroutine call. The list of move operations must be terminated with a zero.

Module Name: OUTFIL

Purpose:

To provide an interface between programs that write files and the actual read, write, and control routines. The subroutine writes fixed block type data records and performs most of the actual file support operations through the QSAM routines.

There are five entry points to the module:

OUTFIL -	To acquire up to four files for subsequent output operations.
INOUT -	To open a given file for output.
PUTCHA -	To write the next sequential record to the file.
CLOUT -	To close the given file and write out any remaining data blocks.
FREOUT -	To release the file. Subsequent output operations will require another call to OUTFIL.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Subroutines Used:

BTD, CHKFIL, EQUC, FINDC, IGC, LAND, LCOMC, MOVEC, PDNCHK, QSAM, SERCOM, SETC, TIME

Logical I/O Units:

SERCOM - Error messages

Description:

These routines will write sequential fixed-block records onto a tape or disk file. If the file is on disc, it may be labelled or unlabelled. If the file is on tape, all the control operations necessary to position the tape to the desired DSN and to set the necessary blocking information is handled by OUTFIL.

Entry Point: OUTFIL

Module Name: OUTFIL

Purpose:

To set up output files for later use by INOUT, PUTCHA, and CLOUT.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL OUTFIL(DSR,PDNAME,VOLUME,DSNAME,FILENO,LRECL,BLKFAC,
            &RC4)
```

Parameters:

DSR	An INTEGER*4 file reference number (1-4). This number serves as an index to the output unit for all subsequent file access.
PDNAME	The LOGICAL*1 pseudo-device name if the file is to be written on tape, or blank if the file is to be written on disk. If not blank, the name must be 3 to 16 characters in length and must terminate with a trailing blank.
VOLUME	The LOGICAL*1 6 character volume serial name for the pseudo-device specified by PDNAME, or blank for disc files.
DSNAME	The LOGICAL*1 DSN if the file is to be written on tape, or the file name if the file is to be written on disk. The name must be 1 - 17 characters in length and must terminate with a trailing blank.
FILENO	The INTEGER*4 file designator. For tapes = the file number (or "0" for *EOT*). For disc = "0" for labelled files = "1" for unlabelled files.
LRECL	The INTEGER*4 logical record length, or "0". If zero is used, the record length <u>must</u> be supplied on the INOUT call.

Parameters: (Continued)

BLKFAC	The INTEGER*4 blocking factor, or "0". If zero is used, the blocking factor is chosen to be the truncated value of "28000/LRECL" for tapes or "1" for disc files.

Return Code(s):

&RC4 -	A wide variety of errors that occur in setting up the file. Error returns are generally preceded by a message from OUTFIL or from QSAM.

Description:

If a disk file is specified (as indicated by a blank PDNAME), the file name is checked for validity, an FDUB is acquired, and the file parameters are stored in internal arrays.

If a previously used tape is specified, an FDUB is acquired, the volume name of the tape is checked against the name supplied, the current tape position is determined, and the file parameters are stored in internal arrays.

If a previously unused tape is specified, the same operations described above are performed, but in addition blocking is disabled on the tape (i.e., BLK=OFF).

Entry Point: INOUT

Module Name: OUTFIL

Purpose:

To open a file acquired by OUTFIL for subsequent write operations.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL INOUT(DSR,LRECL,&RC4)
```

Parameters:

DSR	The INTEGER*4 file reference number used in a previous call to OUTFIL.
LRECL	The INTEGER*4 logical record length if not supplied in the OUTFIL entry.

Return Code(s):

&RC4 - Many error resulting from tape positioning, header records, etc.

Description:

If the file is an unlabelled disk file, the file is rewound and opened for writing with the specified logical record length.

If the file is a labelled disk file, the file is rewound and a header record is written. The file is then opened for writing.

If the file is on tape, the tape is positioned to the end of tape if the file number is zero. If the file number is not zero, date checking is turned off, a warning message is printed and the tape is positioned to the specified file number. The tape DSR is controlled for the specified DSN and blocking format, and the file is opened for writing.

Entry Point: PUTCHA

Module Name: OUTFIL

Purpose:

To write the next sequential record to the output file.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL PUTCHA(DSR,DATA,&RC4)
```

Parameters:

DSR	The INTEGER*4 file reference number used in a previous call to OUTFIL.
DATA	The region containing the data record to be written.

Return Code(s):

&RC4 - Error return from QPUT

Description:

The subroutine entry QPUT is called to write the next record.

<u>Entry Point:</u> CLOUT

<u>Module Name:</u> OUTFIL

<u>Purpose:</u>

To close the specified input file and write out any unfilled data blocks.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

```
CALL CLOUT(DSR,&RC4)
```

<u>Parameters:</u>

DSR	The file reference number used in a previous call to the entry OUTFIL.

<u>Return Code(s):</u>

&RC4 - Control error on PDNAME.

<u>Description:</u>

Any unfilled data blocks are written out. If the file is on disc, it is closed. If the file is on tape, a tape mark is written, date checking is turned on, and the file is closed.

Entry Point: FREOUT

Module Name: OUTFIL

Purpose:

To release the file acquired by entry OUTFIL.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL FREOUT(DSR,&RC4)
```

Parameters:

DSR	The file reference number used in a previous call to OUTFIL.

Return Codes:

&RC4 - Invalid DSR number.

Description:

If the file is on disc, a check is made to see if another DSR uses the same DSN. If not, then the FDUB is released. All internal arrays for this DSR are initialized.

If the file is on tape, a check is made to see if another DSR uses the same tape. If not, the FDUB is released and the internal arrays for this DSR are initialized.

<u>Entry Point:</u> PDNCHK

<u>Module Name:</u> PDNCHK

<u>Purpose:</u>

To check a pseudo-device name to see if it a valid MTS name.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u> CALL PDNCHK(PDN,PDNLEN,&RC4)

<u>Parameters:</u>

PDN	A LOGICAL*1 array containing the PDname to be checked.
PDNLEN	The INTEGER*4 length of the name in PDN.

<u>Return Code(s):</u>

&RC4 The PDname is invalid.

<u>Subroutines Used:</u> EQUC, FINDC

<u>Logical I/O Units:</u> None

<u>Description:</u>

The following checks are made:

1) Is PDNLEN < 1
2) Is PDNLEN > 16
3) Does PDN(1) = '*'
4) Does PDN(PDNLEN) = '*'
5) Does PDN contain , ; : ( ) @ + = ' " ? & or blanks

Entry Point: PRNTCK

Module Name: PRNTCK

Purpose:

To check a string for non-printing characters and insert a question mark in place of any that are found.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence: CALL PRNTCK(STRING,LENGTH)

Parameters:

STRING	The LOGICAL*1 array containing the string to check.
LENGTH	The INTEGER*4 length of the string in STRING.

Description:

Printing characters are assumed to be one of the set of EBCDIC characters with DECIMAL values:

64,75-80,90-97,107-111,122-127,129-137,139,145-155,
162-170,173,186,189,193-201,209-217,226-233,240-249

This routine may be used to replace any non-printing characters in a string before printing it out. It is useful for echoing back user input that may contain bad characters that were entered accidentally.

Module Name: QSAM (Queued Sequential Access Method)

Purpose:

To read and write blocked records consisting of one or more fixed-length logical records. The blocked input/output routines have the following seven FORTRAN entry points:

QGTUCB	To acquire a file or device
QOPEN	To open the file or device for reading or writing
QGET	To read a logical record
QPUT	To write a logical record
QCLOSE	To close the file or device and write any unfilled blocks
QCNTRL	To perform any valid MTS control operation
QFRUCB	To release the file or device

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Subroutines Used:

GETSPACE, GETFD, GDINFO, READ, FREESPAC, FREEFD, REWIND#, SERCOM, WRITE, CONTROL

Logical I/O Units:

READ	Read blocked records
WRITE	Write blocked records
SERCOM	Error messages

Description:

These routines will read and write blocked input/output records consisting of one or more fixed length logical records. All input/output requests are made for logical records: the routines handle record blocking and deblocking automatically. More that one file or device may be handled at one time. These routines are intended for use with magnetic tapes and tapes must be mounted with BLK=OFF. The routines are not restricted to tape usage, however, and may be used with disc files, or with other devices.

Many internal error messages can be generated. Each of these has the form:

```
"device name": <message text>
```

In addition, if a return code greater than zero is encountered in the CONTROL or WRITE routines, or if a return code greater than four is encountered in the READ routine, then the MTS error message associated with this return code is also printed if this message is available. See the MTS Manual, Volume 3, for a description of the I/O error return codes.

Entry Point: QGTUCB

Module Name: QSAM

Purpose:

To acquire a file or device which will be used by the I/O routines. A table of control information for the file or device is generated.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

CALL QGTUCB(NAME,UCBPTR,&RC4)

Parameters:

NAME	The 17 character (max) file or pseudo-device name to be used for the I/O operations terminated by a trailing blank.
UCBPTR	An Integer*4 pointer to the UCB for this fdname that is used by the remaining routines as an index to this device.

Return Code(s):

&RC4 Invalid file or device name

Description:

A chain of all UCB's acquired thus far is searched to see if this file or device has been acquired before. If so, the UCB pointer is returned immediately. Otherwise, a UCB is built and added to the chain, and a pointer to it is returned. The routines GETFD and GDINFO are called and pertinent information is stored in the UCB. The comparison is performed for the full name given. That is, F and F(1,10) are considered to be different files or devices.

Entry Point:	QOPEN
Module Name:	QSAM

Purpose:

To prepare a file or device which has been acquired by QGTUCB for blocked input/output operations.

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL QOPEN(UCBPTR,KEY,BLKFAC,LRECL,&RC4)
```

Parameters:

UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB
KEY	An INTEGER*4 variable that indicates whether the information is to be read or written: 1 Information is to be written 2 Information is to be read
BLKFAC	The INTEGER*4 maximum number of logical records per physical record
LRECL	The INTEGER*4 length of each logical record in bytes

Return Code(s):

&RC4	File or device is already open Incorrect READ/WRITE parameter specification Maximum record length rejected by tape DSR

Description:

The file or device specified by UCBPTR is checked to determine if it has been opened by previous calls to QOPEN. The read/write parameter KEY is checked for validity. The block size of the blocked record is computed as BLKFAC*LRECL and a buffer is acquired for this record.

<u>Entry Point:</u> QGET

<u>Module Name:</u> QSAM

<u>Purpose:</u>

To acquire the next sequential logical record from the file or device opened as an input file via QOPEN.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Calling Sequence:</u>

CALL QGET(AREA,UCBPTR,&RC4,&RC8)

<u>Parameters:</u>

AREA	The input area where the next logical record will be stored
UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB

<u>Return Code(s):</u>

&RC4	End-of-file detected on input file or device
&RC8	The file or device has not been opened for input Device used after an end-of-file Input is longer than the maximum specified Return code > 4 from READ

<u>Description:</u>

Physical records are read from the input file or device as required. Each physical record is broken into one or more logical records of the length specified in the QOPEN call. The last logical record in a physical record may actually be shorter than the length of the logical record. In that case, it is padded to the proper length with blanks. If there are no more logical records, the input area is filled with X'FF'.

<u>Entry Point:</u> QPUT

<u>Module Name:</u> QSAM

<u>Purpose:</u>

To write the next sequential logical record to the file or device opened as an output file via QOPEN.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> 370 Assembler

<u>Calling Sequence:</u>

CALL QPUT(AREA,UCBPTR,&RC4)

<u>Parameters:</u>

AREA	The output area where the next logical record is stored
UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB

<u>Return Code(s):</u>

&RC4 The file or device has not been opened for output
Return code > 0 from WRITE

<u>Description:</u>

Each logical record presented by a call to QPUT is placed into an output buffer. When the buffer is filled, it is written out as one physical record. All physical records will contain the maximum number of logical records specified by the call to QOPEN except the last, which will be truncated if it is only partially filled when QCLOSE is called.

Entry Point: QCLOSE

Module Name: QSAM

Purpose:

To terminate blocked input/output operations on the file or device opened by a call to QOPEN. If the file or device was used for output, and a partially filled buffer of logical records is present, the truncated buffer is written out as part of the closing procedure.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
CALL QCLOSE(UCBPTR)
CALL QCLOSE(0)
```

Parameters:

UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB. If a zero is specified for UCBPTR, then all currently open files or devices are closed.

Description:

If the file or device was used for output and a partial buffer of logical records for it is present, this buffer is written out as a truncated physical record. All information in the UCB is reset to the normal state of an unopened file or device which is then available for further use and can be reopened or positioned.

Note that no tape mark is written when an output file is closed. If a tape is repositioned, a tape mark will be automatically be written by the tape DSR.

Entry Point:	QCNTRL
Module Name:	QSAM

Purpose:

To perform any valid MTS control command for the file or device specified. For magnetic tapes, a complete presentation of these commands is presented in MTS Manual, Volume 4 "TERMINALS AND TAPES".

Location:	HSRI:LIBRARY
Source Language:	370 Assembler

Calling Sequence:

```
CALL QCNTRL(COMMND,LEN,UCBPTR,&RC4)
```

Parameters:

COMMND	An array containing the control command
LEN	The INTEGER*2 length of the control command in COMMND
UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB.

Return Code(s):

&RC4 The file or device is open and cannot be CONTROLled
Improper control operation
No control entry or illegal FDUB pointer
Return code > 0 from CONTROL
Unable to rewind device
Device has no type and cannot be CONTROLled
Device has no FDUB and cannot be CONTROLled

Description:

If REW is specified, then the routine REWIND# is called to rewind the file or device. For all other control command, the routine CONTROL is called to perform the specified operation.

Entry Point: QFRUCB

Module Name: QSAM

Purpose:

To free a file or device which has been acquired via QGTUCB.

Location: HSRI:LIBRARY

Source Language: 370 Assembler

Calling Sequence:

```
CALL QFRUCB(UCBPTR)
CALL QFRUCB(0)
```

Parameters:

UCBPTR	The INTEGER*4 UCB pointer returned by QGTUCB. If a zero is specified for UCBPTR, then all currently open files or devices are released.

Description:

The chain of all UCB's acquired is searched for the UCB specified by UCBPTR. If it is found, the UCB is deleted from the chain and released. Any subsequent operations on this file or device must be preceded by a call to QGTUCB in order to reallocate its UCB.

Entry Point: RBTD

Module Name: RBTD

Purpose:

To convert an integer number and a corresponding implied number of decimal places to a character representation.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence: CALL RBTD(INTEGR,ARRAY,FLDWTH,NUMDEC)

Parameters:

INTEGR	The INTEGER*4 value to be converted.
ARRAY	The LOGICAL*1 array where the character representation will be placed.
FLDWTH	The INTEGER*4 length of the output field.
NUMDEC	The INTEGER*4 number of implied decimal places for INTEGR.

Subroutines Used: BTD, SETC

Description:

If NUMDEC is zero, INTEGR is converted into the output array with a maximum length of FLDWTH. If the number is too large to fit into the desired output field, the field is filled with asterisks. If NUMDEC is greater than zero, the number with decimal point is written.

Entry Point: READIN

Module Name: READIN

Purpose:

To read input into an array on I/O unit SCARDS with the features of line continuation, upper-case conversion, array length protection, and notification of unit reassignment.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

CALL READIN(STRING,LENGTH,MAXLEN,SOUSW,&RC4,&RC8,&RC12)

Parameters:

STRING	A LOGICAL*1 array of dimension MAXLEN.
LENGTH	A INTEGER*4 variable that contains, on exit from the routine, the length of the input string plus one for the trailing blank added to the end of the string.
MAXLEN	The INTEGER*4 length of STRING.
SOUSW	A LOGICAL*1 switch that is .TRUE. if SCARDS is assigned to the terminal or card reader, and .FALSE. otherwise.

Return Code(s):

&RC4	An end-of-file was encountered with SOUSW = .TRUE.
&RC8	An end-of-file was encountered with SOUSW = .FALSE.
&RC12	The input line length is greater than MAXLEN-1

Subroutines Used:

ADROF, BTD, EQUC, GDINF, LAND, LOR, MOD, MOVEC, RCALL, SETLIO, SETPFX

Logical I/O Units:

SCARDS	Read input string
SERCOM	Error messages

Description:

This subroutine is intended as a general user input. The prefix character for the read is set to "?". The read is made on SCARDS with the modifiers @TRIM, @CASECONV, @MAXLEN, and @NOTIFY. If the input string ends in the continuation character "-", another read is made and the new characters are added on to the end of the previous string beginning at the location of the "-". If the total number of characters read in is greater than MAXLEN-1, a branch is made to RC12. If not, the prefix character is reset, a trailing blank is appended to the string, and a normal return is taken.

If an I/O unit re-assignment occurs GDINF subroutine is called to determine what the assignment is, and SOUSW is set accordingly. The read is then performed as described above.

Entry Point: READNC

Module Name: READIN

Purpose:

To read input into an array on I/O unit SCARDS with the features of upper-case conversion, array length protection, and notification of unit reassignment. The action is the same as the READIN entry with the exception that line continuation is not supported.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL READNC(STRING,LENGTH,MAXLEN,SOUSW,&RC4,&RC8,&RC12)
```

Parameters:

STRING	A LOGICAL*1 array of dimension MAXLEN.
LENGTH	A INTEGER*4 variable that contains, on exit from the routine, the length of the input string plus one for the trailing blank added to the end of the string if no continuation character is present. If there is a continue character at the end of the segment, then LENGTH contains the length of the input string including the continue character.
MAXLEN	The INTEGER*4 length of STRING.
SOUSW	A LOGICAL*1 switch that is .TRUE. if SCARDS is assigned to the terminal or card reader, and .FALSE. otherwise.

Return Code(s):

&RC4	An end-of-file was encountered with SOUSW = .TRUE.
&RC8	An end-of-file was encountered with SOUSW = .FALSE.
&RC12	The input line length is greater than MAXLEN-1

Subroutines Used:

ADROF, BTD, EQUC, GDINF, LAND, LOR, MOD, MOVEC, RCALL, SETLIO, SETPFX

Logical I/O Units:

SCARDS	Read input string
SERCOM	Error messages

Description:

This entry performs the same function as READIN except for line continuation. A trailing dash "-" is ignored by the routine and treated as any other character. If a continue character is found at the end of the line, no trailing blank is appended to the line.

This routine is intended for use in dynamic dimensioning applications where an very long string is read in segments and a buffer array of unknown length must be generated dynamically to hold the information. In such a case, the line continuation must be performed outside of READIN as part of the dimensioning process.

Entry Point:	REPMSG
Module Name:	REPMSG

Purpose:

To print a HELP explanation message for CMDSCN and KEYSCN.

Location:	HSRI:LIBRARY
Source Language:	FORTRAN
Calling Sequence:	CALL REPMSG
Subroutines Used:	SPRINT

Logical I/O Units:

SPRINT - Message output

Entry Point: SHFTST

Module Name: SHFTST

Purpose:

To shift portions of a string to the right or to the left.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL SHFTST(STRING,LENGTH,MAXLEN,START,SHIFT,&RC4)
```

Parameters:

STRING	The LOGICAL*1 array in which the character shift is to be performed.
LENGTH	The INTEGER*4 length of the character string in array STRING.
MAXLEN	The INTEGER*4 length of the array STRING.
START	The INTEGER*4 location where the shift is to start.
SHIFT	The INTEGER*4 number of characters to shift. SHIFT < 0 - Shift left SHIFT = 0 - No shift SHIFT > 0 - Shift right

Return Code(s):

&RC4 - The requested shift would would put part of the string outside of the boundaries of STRING.

Subroutines Used: SETC

Description:

The substring beginning at START and ending at LENGTH is shifted right or left by the number of characters indicated in SHIFT. If the shift is made to the left (i.e., SHIFT < 0) then existing characters in STRING are written over. If the shift is made to the right (i.e., SHIFT > 0) then the portion of the array that is vacated is filled with blanks.

Example(s):

The call SHFTST(STRING,22,80,11,-5,&900) would change the string

```
       This is a test string.
into
       This test string.
```

<u>Entry Point:</u> SLIST

<u>Module Name:</u> SLIST

<u>Purpose:</u>

To decode a character string into elements using the comma as a delimiter.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u>

```
CALL SLIST(STRING,START,LENGTH,NUMBER,ARRAY,MAX,&RC4)
```

<u>Parameters:</u>

STRING	The LOGICAL*1 character string to be decoded.
START	The INTEGER*4 location in STRING where decoding is to begin.
LENGTH	The INTEGER*4 length of the string to decode.
NUMBER	The INTEGER*4 number of elements found in STRING.
ARRAY	An INTEGER*4 array of dimensions (2,MAX) where: ARRAY(1,J) = The location in STRING where element number J starts. ARRAY(2,J) = The length of element number J.
MAX	The INTEGER*4 maximum number of elements that may be specified in STRING.

<u>Return Code(s):</u>

&RC4 - The number of parameters specified is greater than MAX.

<u>Subroutines Used:</u> FINDC

<u>Description:</u>

The array STRING is scanned from START to START+LENGTH-1 for the occurrence of a comma. If the length of the element found is zero (i.e., two sequential commas) then the element is treated as valid, but ARRAY(1,J) and ARRAY(2,J) are set to ZERO. This permits the entry of default element specifications. Scanning continues until the end of the string is reached, or until too many elements have been specified. The string need not terminate with a comma to delimit the last element.

Entry Point:	SPLCHK
Module Name:	SPLCHK

Purpose:

To provide the spelling check and error replacement function for the KEYSCN and CMDSCN subroutines.

Location:	HSRI:LIBRARY
Source Language:	FORTRAN

Calling Sequence:

```
CALL SPLCHK(TSTNAM,TSTLEN,NWRDS,WORD,WRDPAR,WRDNUM,
           &RC4,&RC8)
```

Parameters:

TSTNAM	The LOGICAL*1 array containing the test word.
TSTLEN	The INTEGER*4 length of the test word.
NWRDS	The INTEGER*4 number of words in WORD.
WORD	The LOGICAL*1 array containing the 8-character left-justified valid names.
WRDPAR	The INTEGER*2 array of word lengths of dimension (2,NWRDS) used in CMDSCN and KEYSCN.
WRDNUM	The INTEGER*4 number of the word in WORD that TSTNAM is a misspelling of.

Return Code(s):

&RC4 -	The test word is not a misspelling of any valid word contained in WORD.
&RC8 -	The error replacement request on GUSRIN was CANCELled.

Subroutines Used:

EQUC, FINDST, GUINFO, GUSRIN, IGC, LCOMC, LSTPAR,

MOVEC, PRNTCK, SERCOM, SPELCK, SPRINT

Logical I/O Units:

SERCOM - Error messages

SPRINT - User prompts.

Restriction(s):

The routine uses a LOGICAL*1 140-character COMMON area named /BUF/ for a work array.

Description:

The test word supplied in TSTNAM is checked against the list of valid words in WORD for a possible misspelling using the MTS spelling check routine SPELCK. If a candidate is found, the user is prompted for confirmation of the correct value.

<u>Entry Point:</u> TIMDAT

<u>Module Name:</u> TIMDAT

<u>Purpose:</u>

To generate and print a line containing the current date and time.

<u>Location:</u> HSRI:LIBRARY

<u>Source Language:</u> FORTRAN

<u>Calling Sequence:</u> CALL TIMDAT(UNIT,&RC4)

<u>Parameters:</u>

UNIT	The INTEGER*4 logical I/O unit number.

<u>Return Code(s):</u>

&RC4 - Error return from EWRITE

<u>Subroutines Used:</u> EWRITE, MOVEC, SPRINT, TIME

<u>Logical I/O Units:</u>

SPRINT - Time/date string for UNIT = -1

<u>Restrictions:</u>

The subroutine uses a LOGICAL*1 140-byte common area named /BUF/ to hold the TIME/DATE string.

<u>Description:</u>

Time is called to generate the required output string. If UNIT = -1, then the string is written out on SPRINT. If UNIT = 1-19 then the string is written on the specified unit using EWRITE. For all other values of UNIT, a return is made with the string left in COMMON /BUF/.

<u>Example(s):</u>

```
     Date: Apr 19, 1983  at  14:53:06
```

Entry Point: VLCHEK

Module Name: VLCHEK

Purpose:

To check a list of variable numbers for accuracy in terms of syntax errors as well as valid variable numbers.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL VLCHEK(STRING,FIRST,LAST,NUMBER,ERRCOL,MODE,&RC4)
```

Parameters:

STRING	The LOGICAL*1 character string containing the variable list to be checked.
FIRST	The INTEGER*4 location in STRING where the list begins.
LAST	The INTEGER*4 location in STRING where the list ends.
NUMBER	The INTEGER*4 number of distinct values in the list.
ERRCOL	The INTEGER*4 location of a syntax error. Valid only when RC4 is taken.
MODE	An INTEGER*4 switch to control variable checking. 0 - Check if variable exists 1 - Check if variable exists and is numeric.

Return Code(s):

&RC4 - Syntax error
    Number is too large for INTEGER*4 representation
    Too many values specified (NUMBER > 32767)
    Non-numeric character in number.

Subroutines Used:

BTD, CHKVAR, DTB, EQUC, FINDC, IGC, MOVEC, SERCOM

<u>Logical I/O Units:</u>

SERCOM - Error messages

<u>Restriction(s):</u>

This routine uses a 140-byte LOGICAL*1 COMMON area named /BUF/ and relies on an on-line dictionary for variable checking. These restrictions make it of use only in the ADAAS program.

<u>Description:</u>

The list is syntax checked and each number is converted into binary. The on-line dictionary is accessed to make sure each variable is valid for the data set and the total number of variables in the list is returned.

<u>Example:</u>

The routine expects a list of numbers consisting of single values and ranges separated by commas. Ranges are indicated by two values joined with a dash. For example:

```
     1,5,7-9,12-45,1025,2356-2359
```

Entry Point: VLIST

Module Name: VLIST

Purpose:

To convert a list of numbers in character format into an array of binary values corresponding to elements in the list.

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL VLIST(STRING,FIRST,LAST,NUMBER,NUMLST,MAXNUM,
           ERRCOL,&RC4)
```

Parameters:

STRING	The LOGICAL*1 character string containing the variable list to be converted.
FIRST	The INTEGER*4 location in STRING where the list begins.
LAST	The INTEGER*4 location in STRING where the list ends.
NUMBER	The INTEGER*4 number of distinct values in the list.
NUMLST	The INTEGER*4 array of dimension MAXNUM where the binary values will be stored.
MAXNUM	The INTEGER*4 maximum number of values permitted.
ERRCOL	The INTEGER*4 location of a syntax error. Valid only when RC4 is taken.

Return Code(s):

&RC4 - Syntax error
Number is too large for INTEGER*4 representation.
Too many values specified (NUMBER > MAXNUM)
Non-numeric character in number.

Subroutines Used:

BTD, DTB, EQUC, FINDC, IGC, MOVEC, SERCOM

Logical I/O Units:

SERCOM - Error messages

Description:

The list is syntax checked and each number is converted into binary. The routine is used to convert input lists into binary for use by analysis programs.

Example:

The routine expects a list of numbers consisting of single values and ranges separated by commas. Ranges are indicated by two values joined with a dash. For example:

```
1,5,7-9,12-45,1025,2356-2359
```

Entry Point: VRANGE

Module Name: VRANGE

Purpose:

To decode a value and associated range of the form:
VALUE:MIN-MAX

Location: HSRI:LIBRARY

Source Language: FORTRAN

Calling Sequence:

```
CALL VRANGE(STRING,LENGTH,VALUE,MIN,MAX,&RC4)
```

Parameters:

STRING	A Logical*1 array containing the string to be decoded.
LENGTH	The Integer*4 length of the string.
VALUE	The Integer*4 value for VALUE. VALUE is set to zero if the error return is taken, or if the string specifies the value "NONE"
MIN	The Integer*4 value for MIN
MAX	The Integer*4 value for MAX

Return Code(s):

&RC4 Syntax error
Non-numeric character
Number too large for Integer*4 representation
Minimum value greater than maximum

Subroutines Used: DTB, EQUC, FINDC, IGC, LCOMC, SERCOM

Logical I/O Units:

SERCOM Error messages

Description:

Five possible input configurations are possible. They are decoded as follows:

STRING	VALUE	MIN	MAX
"xx:yy-zz"	xx	yy	zz
"xx:yy"	xx	yy	yy
"xx"	xx	*	*
"NONE"	0	*	*
"none"	0	*	*

The asterisk indicates that the values of MIN and MAX are not changed by VRANGE for these input strings.

If an error occurs, an error message and the column location of the error is printed on SERCOM. The value of VARNUM is set to zero, but the values of MIN and MAX may have been changed.