MUSIC/SP User's Reference Guide

Part 5 - Chapter 9 (UR_P5.PS)

Chapter 9. System Subroutines

Chapter 9. System Subroutines

Overview

This chapter describes the MUSIC system subroutines. Many other subroutines are also available to the MUSIC user such as the mathematical functions SQRT, and so forth.

First, a listing of the routines in functional groups will be presented. Next, an alphabetical arrangement of all these subroutines with detailed usage descriptions of each is presented.

Functional Summary of Subroutines

Scheduling Actions and Controlling Jobs

This group of subroutines cause a MUSIC activity to be scheduled after the current job ends.

NONCAN	Make a program noncancelable.
CANCAN	Remove the effect of a call to NONCAN.
NXTCMD	Schedule a command to be run next using multi-tasking or program chaining.
NXTPGM	Schedule a program to be run next using program chaining.
PARM	Pick up the information from a /PARM control statement.
SAVREQ	Schedule a /SV command next.
SIGNOF	Schedule a /OFF next.
NSGNOF	Removes the request to schedule a /OFF command next.
SYSMSG	Suppress system information messages until after the scheduled activity is done.
EOJ	Terminate the current job and sets a return code.
GETRET	Get return code.

Time, Date and User Information

These routines can obtain the time and date in various formats. Information about the userid running the job and the workstation type is also available. Routines are provided to give elapsed job time. A delay routine can be used to re-activate a job after a given amount of time.

CMDRET	Retrieves command strings.
CMDSTO	Stores command strings.
CODON	Determines if a userid is signed on to MUSIC.
DATCON	Converts date data from one format to another.
DATCN2	Similar to DATCON with a more thorough analysis of the argument list.
DELAY	Specify time delay before job is re-activated.
GETID	Gets ownership id from a file name or data set name.
STATUS	Returns the time of day, current date, service units used, etc.
TIMCON	Converts time data from one format to another.
TIMDAT	Gets time of day (by TSTIME) and date (by TSDATE).
TIMOFF	Job time is displayed.
TIMON	Reset time counter used in TIMOFF.
TSDATE	Give current date.
TSTIME	Give current time of day or execution time.
TSUSER	Provide userid and workstation type.

Dynamic Access

These routines allow you to read and write files from FORTRAN without having to pre-define the file names on /FILE statements. The files are opened and closed dynamically.

OPNFIL	Open a file.
CLSFIL	Close a file.
SETINF	Specify information for a new file.
FILMSG	Get error description text.
PURGE	Delete a file.
FRSTOR	Frees main storage for use via an array.
GTSTOR	Gets main storage for use via an array.
QFOPEN	Opens a file and defines buffer area
QFCLOS	Closes a file and releases buffer
QFREAD	Reads next logical record
QFBKRD	Reads previous logical record
QFRBA	Sets RBA for next read
QFREW	Rewinds the file

SYSIN Read Routines

These routines allow you to dynamically specify which file to read from the Save Library. They also allow you to inspect where the SYSIN (unit 5) data is coming from and issue messages.

SYSINR	Dynamically open a file for reading.
SYSINM	Give message identifying file and line number.
SYSINE	Inspect error return codes for sysin.
SYSINL	Inspect SYSIN nesting stack.

Block Transfer

These routines allow reading and writing of large blocks to disk files, bypassing the logical blocking and buffering operations. A large number of separate logical files are allowed.

GULPDF	Defines file characteristics for GULP routines.
GULPRD	Read in GULP mode.
GULPWR	Write in GULP mode.

FORTRAN I/O

These routines are used in conjunction with FORTRAN I/O. They can re-read the last logical record, do I/O to an in-core buffer, allow mixed free-and fixed-format I/O, and close a direct access file.

MCNCAT	Replace VS Fortran's CNCAT# concatenation routine.
REREAD	Re-read last logical record.
CORE	DO I/O to buffer in main storage.
XGCON	Allow mixed free and formatted I/O.
XGCOFF	Turn off XGCON mode.
CLOSDA	Close direct access buffers.
FRMTDA	Erase the contents of a direct-access data set.
BIGBUF	Allow record lengths of greater than 133 bytes to be used.

ASCII Terminals

TPOPEN	Turn off line folding at col 72 on a TTY terminal.
TPCLSE	Turn on line folding at col 72 for TTY terminal.
NOCRLF	Stop automatic return at the end of lines.
CRLF	Re-instate automatic return at end of line.
NOTRIN	Suppress translation of terminal input.
TRIN	Cancel a call to NOTRIN and resume the normal translation of terminal input.

3270-Type Workstations

These routines allow you to control the display of the input area on 3270-type workstations when performing a conversational read.

ADTOXY	Convert a 3270 2-byte screen address to row and column.
KEEPIN	Do not erase the contents of the input area.
CLRIN	Erase the contents of the input area.
NOECHO	Do not display the contents of the input area on the output area.
ECHOIN	Display the contents of the input area on the output area.
NOSHOW	Do not display the characters being entered to the input area.
SHOWIN	Display the characters being entered to the input area.

Workstation Prompting

These routines allow you to remove and restore the conversational read prompt.

NPRMPT	Remove prompting entirely
PROMPT	Do prompting again

Workstation Features

These routines can be used to set TABS, turn on and off line folding, nonstop mode, and type element return functions.

TABS	Informs MUSIC of desired TAB settings.
NOPAUS	Set nonstop mode. Used mainly for 1050 terminals.
PAUSE	Turn off non-stop mode. Used mainly for 1050 terminals.
TEXTLC	Lower case letters input during execution are to be preserved as such.
TEXTUC	Lower case letters input during execution are to be converted to their upper case equivalents.

Controlling Output

STOPSK	Stop /SKIP operation requested by the user.

Controlling Tape

BACKSP	Do a backspace operation on a tape file defined by a ddname.

Manipulating Bits

Routines that work on the 8 bits of a byte. Each bit of the byte has the two possible values of ON or OFF.

BITON	Turn bit on.
BITOFF	Turn bit off.
BITFLP	Reverse a bit.
TBIT	Test if bit is on or off.

Character Translation

CTRAN	Translate each occurrence of one character to another.
FIXSCR	Translates hex characters to blanks.
TOLC	Translates a string of characters to lower case.
TOUC	Translates a string of characters to upper case.
TRANSL	Modify characters using a translate table.

Moving Bytes and Words

These routines get or move storage on a byte or word basis. One routine can set many bytes to zero.

BYTE	Get first byte of arg.
FILL	Fill storage with a specified character.
HWORD	Get first 2 bytes of arg.
LMOVE	Move bytes from one place to another.
LBMOVE	Same as LMOVE but on any byte boundary.
MOVE	Move words in storage.
ZERO	Set storage to zeros.

Converting Character Strings

C2D	Convert a numeric character string to a decimal value (double precision).
C2I	Convert a numeric character string to an integer value.
C2R	Convert numeric character string to a decimal value (single precision).
C2X	Convert a character string to hexadecimal.
DECN	Convert a numeric string to an integer value.
DECOUT	Convert an integer value to a character string, similar to Fortran I format.
I2C	Convert an integer value to a character string.
I2X	Convert an integer to a hexadecimal string.
X2C	Convert a hexadecimal string to characters.
X2I	Convert a hexadecimal string to an integer value.
MA2E	Convert from Ascii to EBCDIC (1-to-1 mapping for all 256 characters). Same calling sequence as A2E.
ME2A	Convert from EBCDIC to Ascii (also 1-to-1). Same calling sequence as E2A.
TBAS64	Encode data using Base64 method (MIME standard).
FBAS64	Decode Base64 data.
B64TXT	Processes Base64 data: decode, separate into records by CRLF, and convert from Ascii to EBCDIC. Record input and output is done by calling user-supplied subroutines. This routine could be used by MAIL to display MIME text.
UUENC	Encode data using the UUENCODE method. Similar to Base64, but uses different character table and does not remember the exact original length.
UUDEC	Decode UUENCODE'd data.

Manipulating Strings

ABBREV	Test whether a string is a true abbreviation for a keyword.
CENTER	Center a string.
GETOP	Get the parameters from a character string.
LJUST	Left justify a string.
LN	Gets the length of a character string.
NXWORD	Get the next word from a string.
PROCOP	Process an option item in the format abc(xyz).
RJUST	Right justify a string.
RVRS	Reverse the order of characters in a string.
SEP	Separate out fields separated by 1 or more blanks/commas.
WORD	Separate a string into words.

Character Searching

FNDALL	Find all specified *reference* characters in a string.
FNDCHR	Find the first *reference* character in a string.
LOCATE	This routine finds the first occurrence of one string within another string.
VERALL	Verify all characters of a string are within a set of reference characters and returns all mismatches.
VERIFY	Verify all characters of a string are within a set of reference characters and returns the first mismatch.

Comparing Bytes and Words

Routines to compare numbers or character strings for equality (and so forth).

EQUAL	Compare strings for equality.
EQUALB	Compare byte strings for equality.
LCOMP	Compare strings.
LBCOMP	Compare byte strings for equality.

Logical Operations

Perform logical operations and shift bits within words.

LAND	Logical AND.
LOR	Logical OR.
LXOR	Logical exclusive OR.
LCOMPL	Logical complement.
LSHFTL	Do left shift operation.
LSHFTR	Do right shift operation.

Sorting

See "Sorting Routines" in *Chapter 10. Utilities* for larger disk sort operations.)

DSORT	Generalized disk sort subroutine, see *Chapter 10. Utilities*.
SSORT	Sort arrays in main storage.
SRTMUS	Similar to DSORT, see *Chapter 10. Utilities*

Generating Random Numbers

This routine, with its several function calls, allows for various types of random number generation.

RSTART	Random number generation

PANEL Support

MODFLD	Redefine the attributes of modifiable fields by field number of screens designed using PANEL.
MODOPT	Modify panel options previously defined using the PANEL subsystem.
PANFLD	Query information about PANEL fields.

Debugging

DEBUG	Invokes the Debug Facility at various points in your program.

TCP/IP Sockets

CARGCALL	Guarantees that arguments are passed correctly for C programming.

Index Text Searching (ITS)

ITSFID	Initialization/dynamic call
ITSFIW	Initialization/workarea call
ITSFOP	Open a search set
ITSFSS	Search string
ITSFOR	Order result list
ITSFRE	Retrieve results
ITSFCL	Close search

Miscellaneous

LOCNAM	Change an output file name to make it local to the current directory (if necessary).

Subroutines Listed Alphabetically

ABBREV

This subroutine tests whether a string is a true abbreviation for a keyword.

Calling Sequence: k=ABBREV(a,len,kw,minl,maxl)

Arguments:

a is a true abbreviation.

len is the length in bytes.

kw is the keyword.

minl is the minimum length of the abbreviation.

maxl is the maximum length of the abbreviation.

ABBREV returns k=1 if *a* (of length *len* bytes) is a true abbreviation for *kw* (minimum abbreviation length *minl,* total length *maxl),* otherwise k=0. Should have 1 <= minl <= maxl <= 256.

```
        return k=1 if:       len>0
                       and   len>=minl
                       and   len<=maxl
                       and   equal(a,kw,len)
```

See also: NXWORD, PROCOP, and GETOP.

ADTOXY

This subroutines converts a 3270 2-byte screen address to row and column numbers.

Calling Sequence: CALL ADTOXY(addr,width,row,column).

Arguments:

addr (input) 2-byte screen addr, either 12-bit coded form or 14-bit uncoded form.

width (input) screen width (e.g. 80 or 132).

row (output) row number (>=1).

column (output) column number (1 to width).

BACKSP

This routine preforms a backspace operation on a tape file defined by a ddname. It can be used by a COBOL or PL/I program to add data to the end of an existing tape file. Refer to the discussion on /FILE for tape.

Calling Sequence: CALL BACKSP('ddname ')

BIGBUF

(Fortran G1 only) This subroutine is used to increase the size of the maximum record length handled by FORTRAN sequential I/Os. If this subroutine is not called, the maximum is 133 bytes.

To allow record lengths of more than 133, call BIGBUF and supply an array of length *n* bytes. The system uses a work area buffer for all input/output except direct access. BIGBUF is normally called once at the beginning of the job, before any input/output is done.

Calling Sequence: CALL BIGBUF(buffer,n)

Arguments:

buffer — is the name of the array that is used as an I/O buffer. Since the array must start on a fullword boundary, it is convenient to use an array of type INTEGER*4 or REAL*4.

n — is any number 133 or more indicating the record length. Regardless of the limit defined by BIGBUF, the maximum record length in a file may be subject to other limitations. For further details see *Chapter 4. File System and I/O Interface* of this guide.

Example:

The following FORTRAN program writes three 400-byte records to a temporary file, then reads the records back and displays them. If BIGBUF were not called, this program would give the error message: IHN212I, END OF RECORD ON UNIT 1.

```
/FILE 1 NAME(&&TEMP) NEW DELETE LRECL(400)
      INTEGER BUF(100),A(80)
      CALL BIGBUF(BUF,400)
      WRITE(1,10) (I,I=1,240)
10    FORMAT(80I5)
      REWIND 1
15    READ(1,10,END=20) A
      WRITE(6,*) A
      GO TO 15
20    STOP
      END
```

BITFLP

This subroutine is identical to BITON except that the referenced bit is *reversed*. That is, a 0 is set to 1, and a 1 is set to 0. This routine is an INTEGER function and must be declared as such in your program.

Calling Sequence: n=BITFLP(a,k)

Arguments:

a — location of the byte.

k — is a number between 0 and 31 indicating the bit position.

BITOFF

This subroutine is identical to BITON except that the referenced bit is set to 0 instead of 1. This routine is an INTEGER function and must be declared as such in your program.

Calling Sequence: n=BITOFF(a,k)

Arguments:

a location of the byte.

k is a number between 0 and 31 indicating the bit position.

BITON

This FORTRAN INTEGER function is used in conjunction with TBIT, BITOFF, and BITFLP. It returns the current value of bit *k* (k=0,1,2,...) of location *a*, and sets that bit to 1. This routine is an INTEGER function and must be declared as such in your program.

Calling Sequence: n=BITON(a,k)

Arguments:

a location of the byte.

k is a number between 0 and 31 indicating the bit position. For example, if *k* is 24, the value of the first bit of the fourth byte of *a* is returned and that bit is set to 1. The first bit position is referred to as bit 0. If *k* is negative, a value of zero is returned and the bit is not changed.

BYTE

This subroutine is a FORTRAN INTEGER function which returns the value (0 to 255) of the first byte of the argument *x*. This routine is an INTEGER function and must be declared as such in your program.

Calling Sequence: n=BYTE(x)

B64TXT

This routine processes BASE64 data, as in mime mail data, decodes it, separates it into records, and converts it from ASCII to EBCDIC. Records are assumed to be separated by ASCII CRLF (X'0D0A'). Input records (max 80 bytes each) are read by calling user-supplied routine B64TRD. Output records are written by calling user-supplied routine B64TWR. (this routine is reentrant.)

In the input BASE64 data, blanks and line breaks are ignored. Also, control characters (less than x'40') are also ignored. An input quartet (group of 4 encoded bytes) may be split by these characters or line breaks. If invalid characters are found in the BASE64 data, the corresponding output triples are replaced by "???".

Routines used: FBAS64 - decode base64 data; MA2E translate from 8-bit ascii to ebcdic.

Calling Sequence: CALL B64TXT(work,wrklen,mxolen,argrd,argwr,numout,retcod)

Arguments:

work — a work area, on a doubleword boundary.

wrklen — (input) length of the work area. Should be at least 400 + mxolen.

mxolen — (input) maximum length of output records. when looking for crlf's to separate the records, records longer than this are truncated. if mxolen<1, vaule 1 is used.

argrd — (input) this argument is passed to B64TRD routine.

argwr — (input) this argument is passed to B64TWR routine.

numout — (output) number of output records.

retcod — (output) return code:
0 normal, no errors.
1 work area is too small.
2 terminated by error from b64trd routine.
3 terminated by error from b64twr routine.
4 1 or more bad characters were found in the base64 data. the output text may be incorrect.
5 1 or more output records were truncated.
6 both retcod conditions 4 and 5.

The following is the calling sequences for user-supplied record read/write routines:

Calling Sequence: CALL B64TRD(argrd,rec,maxlen,len,k)

Arguments:

argrd — same as arg passed to B64TXT.
rec — buffer to receive input record.
maxlen — max length of input record (= buffer size). currently maxlen is 80.
len — (output) actual length of record (0 to maxlen). B64TXT skips 0-length input records.
k — (output) 0=normal, 1=eof, 2=error (B64TXT stops). Note: if an EOF quartet (one with the special pad character "=") is found in the input, B64TXT stops before eof is received from b64trd.

Calling Sequence: CALL B64TWR(argwr,rec,len,k)

Arguments:

argwr — same as arg passed to b64txt.
rec — record to be put out.
len — length of the record (0 or more).
k — (output) 0=normal, 1=error (b64txt stops).

CANCAN

A call to this routine removes the effect of a previous call to the NONCAN routine. It makes the program cancelable by the /CANCEL command. A call to CANCAN does not take effect until after the next

conversational read or call to CLSOUT.

Calling Sequence: CALL CANCAN

CARGCALL

This subroutine is intended for C programmers calling TCP/IP socket routines. When called before the first socket routine, it guarantees that the arguments are passed in the manner C requires.

Calling Sequence: CALL CARGCALL();

CENTER

This routine centers a substring, within a 1-to 256-character string, defined as the first non blank character from the left and the last non blank character. The substring is then placed at the center of the string.

Calling Sequence: CALL CENTER(string,strlen,pad)

Arguments:

string	is a 1-to 256-character string to be centered.
strlen	is the length of the string, an integer in the range of 1-256. When strlen=0 no action is taken by CENTER.
pad	all leading and trailing blanks in the string are converted to the pad character, after the text has been centered.

Example:

Given that:

```
STRING --> 'bb12345bbbbb'
STRLEN --> 12
PAD    --> 'b'

CALL CENTER(STRING,STRLEN,PAD)
```

Returns:

```
STRING --> 'bbb12345bbbb'
STRLEN --> 8
```

CLOSDA

(Fortran G1 only) A call to this subroutine *closes* a direct access file specified by *unit.* This close operation causes the contents of buffers to be written out on disk. Also, a direct access read following a call to CLOSDA actually reads from disk, rather than from a buffer in main storage. This is important for applications where several programs share a direct-access file. You may still use the file for further direct-access operations. Note that direct-access data sets may not be closed by the system unless the job terminates

normally.

Calling Sequence: CALL CLOSDA(unit)

CLRIN

A call to this subroutine cancels the effect of a previous call to the KEEPIN subroutine.

Calling Sequence: CALL CLRIN

CLSFIL

Dynamically close a file. Refer to the section "Dynamic Access to Files" for a description of this routine.

CMDRET

This routine retrieves command strings from a circular buffer of previously stored commands. This works in the same way as the store/retrieve (F12) when in *Go. The CMDSTO is used to store the commands strings in the buffer.

Calling Sequence: CALL CMDRET(cmd,cmdlen,maxlen,cmdbuf)

Arguments:

cmd	a 1-126 character string returned from the command buffer
cmdlen	the length of the string CMD is returned.
maxlen	the maximum length of the string CMD (1-126)
cmdbuf	a 256 character string to be used as the command buffer.

Notes:

1. Only the first 128 bytes of CMDBUF are used, the other 128 bytes are used as a temporary buffer.
2. Successive entries of the same CMD string will not be stored.
3. This routine is re-entrant.

CMDSTO

This routine stores a command string into a circular buffer for later retrieval. This works in the same way as the store/retrieve (F12) when in *Go. The CMDRET is used to retrieve from the commands from the buffer, in the order they were entered.

Calling Sequence: CALL CMDSTO(cmd,cmdlen,maxlen,cmdbuf)

Arguments:

cmd	a 1-126 character string to be placed in the command buffer.
cmdlen	the length of the string CMD (1-126).
maxlen	the maximum length of the string CMD (1-126).
cmdbuf	a 256 character string to be used as the command buffer.

Notes:

1. Only the first 128 bytes of the CMDBUF are used, the other 128 bytes are used as a temporary buffer.
2. Successive entries of the same CMD string will not be stored.
3. This routine is re-entrant.

CODON

This routine reads the terminal control blocks (TCB) and determines if the code passed is signed on to MUSIC and returns the TCB number and the code's terminal mode.

Calling Sequence: CALL CODON(code,tcbno,mode,irc)

Arguments:

code	seven-character user id
tcbno	return terminal (TCB number) 2 character integer
mode	return terminal mode 0= not active 1= *go 2= break 3= conversational 4= spooled input 5= running a job 6= FSIO 7= command execution from a prog
irc	Set to 0 if code is not active, set to 1 if code is active.

CORE

This subroutine for FORTRAN allows reading and writing from a buffer in memory, without physical I/O.

Executing this call statement causes the next formatted or list-directed READ or WRITE to transmit from or to the data area in core beginning at location buffer, and extending length bytes.

Calling Sequence: CALL CORE(buffer,length)

Arguments:

buffer is an array name.

length is the length of the array in bytes. Length must be at least 4 and up to 32000.

Notes:

1. The call to CORE must be executed before each formatted or list-directed READ or WRITE (that is to use an incore buffer) is executed. The unit number in the READ or WRITE statement is ignored. CORE must not be called twice without an intervening formatted or list-directed READ or WRITE statement. The I/O request must not attempt to read or write more than one logical record.

2. Declaring the buffer to be of type LOGICAL *1 has the advantage that each character in the buffer can be accessed by subscripting. The buffer is filled with blanks before any data is written into it.

3. Unformatted READ or WRITE statements, and BACKSPACE or REWIND statements must not be executed between the CALL CORE and the READ or WRITE statement.

4. If REREAD and CORE are used within the same program, REREAD is temporarily disabled when CORE is called.

   For example:

```
      LOGICAL *1 BUFF(80)
    5 CALL REREAD
   10 READ(5,100)X
   20 CALL CORE(BUFF,80)
   30 WRITE(N,102)X
   40 CALL CORE(BUFF,80)
   50 READ(99,103)Y
   60 READ(99,104)Z
```

   x is written into the incore buffer BUFF by statement 30. Statement 50 reads from the incore buffer into *y*. Statement 60 rereads the record read by statement 10.

Example:

```
*Go
list sample
*In progress
/LOAD VSFORT
      DIMENSION A(5)
      LOGICAL *1 BUFF(80),NBUFF(80)
      LOGICAL EQUAL,B1
      DO 1 I=1,5
    1 A(I)=10**I
      WRITE(6,2)A
    2 FORMAT('0',5F12.0)
      CALL CORE(BUFF,80)
      WRITE(N,8)A
    8 FORMAT(5F12.0)
C     NOW COMPRESS MULTIPLE BLANKS IN
C     BUFF TO ONE BLANK, BY COPYING TO NBUFF
      NEW=1
      B1=.FALSE.
```

```
      DO 3 K=1,80
      IF(EQUAL(BUFF(K),' ',1)) GO TO 4
      B1=.FALSE.
    5 NBUFF(NEW)=BUFF(K)
      NEW=NEW+1
      GO TO 3
    4 IF (B1) GO TO 3
      B1=.TRUE.
      GO TO 5
    3 CONTINUE
      NEW=NEW-1
      WRITE(6,7)(NBUFF(I),I=1,NEW)
    7 FORMAT('0',80A1)
      CALL EXIT
      END
*End
*Go
sample
*In progress
MAIN   = 0003DC
003AA8 BYTES USED
EXECUTION BEGINS

         10.        100.       1000.      10000.     100000.

10. 100. 1000. 10000. 100000.
*End
*Go
```

CRLF

A call to this subroutine cancels the effect of a call to the NOCRLF subroutine. This subroutine is for ASCII terminals only.

Calling Sequence: CALL CRLF

CTRAN

This routine locates and translates every occurrence of a specified character to another character. The number of characters changed is returned.

Calling Sequence: CALL CTRAN(string,strlen,oldchr,newchr,kount)

Arguments:

string is character string to be processed of length *strlen.*

strlen is an integer specifying the length of the *string.*

oldchr is the target character in string to be converted to the character specified by *newchr.*

newchr is the new character for which each occurrence of *oldchr* will be swapped in *string.*

kount is an integer returned that reports the number of characters changed as specified.

Example:

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
OLDCHR --> 'b'
NEWCHR --> '*'

CALL CTRAN(STRING,STRLEN,OLDCHR,NEWCHR,KOUNT)
```

Returns:

```
STRING --> '**12345*'
STRLEN --> 8
OLDCHR --> 'b'
NEWCHR --> '*'
KOUNT  --> 3
```

C2D

This subroutine converts a numeric character string to its double precision decimal value. Conversion from character number to a decimal is done by separating the whole and the fractional part. These are then converted to their integer values. *j* and *k* are first converted to long-format floating point. Then *x* is computed from *j.k*10**n.* where *j* is the whole, *k* is the fractional part and *n* is the exponent.

Calling Sequence: CALL C2D(string,strlen,x,irc,badchr,iexp)

string is a 1-to 256-character string on which a substring of character numbers is to be converted to short or long format floating point number, in other words, a real number.

strlen is the length of the string, an integer in the range of 1-256.

x is the real floating point number returned.
x must be declared as REAL*8 (double precision).

irc is the return code describing the conversion.
irc=-1 The string was blank.
irc=0 Conversion was successful.
irc=1 An illegal character was found in the string.
irc=2 Exponent overflow (IEXP>75).
irc=3 Exponent underflow (IEXP<-75).

Note: *x* is set to zero when *irc* is not equal to 0.

badchr A one byte variable where the bad character (non integer character) is returned when *irc=1.*

iexp An integer variable where the exponent of the number being converted is returned when *irc=2* or *irc=3.*

C2I

This routine converts the substring, within a 1-to 256-character string, defined as the first non blank character from the left and the last non blank character, to an integer.

Calling Sequence: CALL C2I(string,strlen,i,irc,badchr)

Arguments:

string — is a 1-to 256-character string on which a substring of character numbers is to be converted to integer.

strlen — is the length of the string, an integer in the range of 1 to 256.

i — is the integer returned.

irc — is the return code describing the conversion.
irc=-1 the string was blank.
irc=0 conversion was successful.
irc=1 an illegal character was found in the string.
irc=2 conversion not possible, number was to large.

Note: *i* is set to zero when *irc* is not equal to 0.

badchr — a one byte variable where the bad character (non-integer character) is returned when *irc=1*.

Example:

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8

CALL C2I(STRING,STRLEN,I,IRC,BADCHR)
```

Returns:

```
STRING --> 'bb12345b'
STRLEN --> 8
I      --> 12345
IRC    --> 0
BADCHR --> 'b'
```

C2R

This subroutine converts a numeric character string to its single precision decimal value. Conversion from character number to a decimal is done by separating the whole and the fractional part. These are then converted to their integer values. *j* and *j* are first converted to long-format floating point. Then *x* is computed from *j.k*10**n.* where *j* is the whole, *k* is the fractional part and *n* is the exponent.

Calling Sequence: CALL C2R(string,strlen,x,irc,badchr,iexp)

Arguments:

string	is a 1 to 256 character string on which a substring of character numbers is to be converted to short or long format floating point number. i.e. a real number.
strlen	is the length of the string, an integer in the range of 1-256.
x	is the real floating point number returned. *x* must be declared as REAL*4 (single precision).
irc	is the return code describing the conversion. irc=-1 The string was blank. irc=0 Conversion was successful. ..text irc=1 An illegal character was found in the string. irc=2 Exponent overflow (IEXP>75) irc=3 Exponent underflow (IEXP<-75) *Note:* that *x* is set to zero when *irc* is not equal to 0.
badchr	A one byte variable where the bad character (non-integer character),is returned when *irc=1*.
iexp	An integer variable where the exponent of the number being converted,is returned when *irc=2* or *irc=3*.

C2X

This subroutine converts (unpacks) a EBCDIC character string to its hexadecimal character representation.

Calling Sequence: CALL C2X(string,strlen,hexstr)

Arguments:

string	is a string of length strlen to be converted to its hexadecimal representation.
strlen	is the length of the string, in the range of 1 to 256. When *strlen* =0 no conversion takes place.
hexstr	is the output string where EBCDIC hexdigits representing the original string are returned. The character string must be at least twice the length of string.

Example:

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8

CALL C2X(STRING,STRLEN,HEXSTR)
```

Returns:

```
STRING --> 'bb12345b'
STRLEN --> 8
HEXSTR --> '4040F1F2F3F4F540'
```

DATCN2

This subroutine is a version of DATCON that does a more thorough analysis of the argument list passed. When an error occurs it passes an error number to the calling program. It does not stop the job, nor does it print error messages. Except for the insertion of the **irc** variable in the calling sequence, DATCN2 parameters are identical to DATCON.

Enhanced error checking includes:

- Checks if too many or too few parameters are passed.

- Except for type 10, all input dates are checked for valid month, that the corresponding number of days in that month is correct, and that the number of days is not greater than a full year. the leap year is taken into account on all verifications.

- For types 4 and 6 the integer year can be expressed as 1989 or 89. "1900" is assumed.

- For type 10, a check is made that it is a packed decimal value only. Since this data was generated by datcn2, datcon, or tsdate, further checking is not required.

Calling Sequence: CALL DATCN2(irc,m,arg1,{arg2,arg3},n,arg1,{arg2,arg3})

Arguments:

irc is the one of the following return codes:

irc=0 no error detected conversion successful
irc=1 input data conversion type is invalid
irc=2 output data conversion type is invalid
irc=3 input data (date) is invalid
irc=4 wrong number of arguments detected.

See DATCON for descriptions of the other arguments.

DATCON

This subroutine is used to convert date data from one format to another. DATCON checks incoming dates for errors and stops the job and writes an error message. In addition, no attempt is made to convert the bogus dates; blanks are returned in character fields and zero values in numeric fields.

Calling Sequence: CALL DATCON(m,arg1,{arg2,arg3},n,arg1,{arg2,arg3})

Arguments:

m is one of the numbers listed below (except 5) representing the input date format.

n is one of the numbers listed below representing the output date format.

1 16 byte date, for example: 'FRI SEP 06, 1989'
2 is an 8 byte date in the form MM/DD/YY, for example: 09/06/89.
3 is an 8 byte date, for example: 06SEP89
4 *arg1* is the integer year (1989); *arg2* is the integer date of the year (249).
5 is the integer day of the week (Sunday is 1, Monday is 2).

6 *arg1* is the integer month (5); *arg2* is the integer day (26); *arg3* is the integer year (1989).
7 is an 8 byte date in the form DD/MM/YY, for example: 26/01/89.
8 is an 8 byte date in the form YYMMDD, for example: 890126.
9 is an 8 byte date in the form DDMMYY, for example: 260189.
10 is a 4 byte date in packed decimal 0077DDDZ format, for example: 0089360C.

DEBUG

Calls to subroutine DEBUG ($SUB:DEBUG.OBJ) can be inserted at various points in a program, to invoke the Debug Facility when execution reaches those points. If Debug is not active, the calls are ignored. The usage in a VS Fortran program is "CALL DEBUG" (no arguments). /INCLUDE $SUB:DEBUG.OBJ must be inserted in the job.

Specify the DEBUG option on /JOB (or on the member name statement after /LOAD XMON). At the beginning of the program, you are presented with the Debug screen. Press F2 to run the program. When a CALL DEBUG is reached in the program, the Debug Facility is invoked and you can do debug operations from the Debug screen. Press F2 to resume execution.

This technique is useful when it is not feasible to trace the entire program or set Debug break points. Note that VS Fortran programs are difficult to trace, because Fortran run-time library modules such as IFYVRENC are normally in the Link Pack Area, which is read-only storage.

DECN

This subroutine converts a numeric character string to an integer value. The string is of length from 1 to 9. A valid string consists of decimal digits, without any leading blanks.

Calling Sequence: CALL DECN(string,len,ivalue,irc)

Arguments:

string is a numeric character string specified by the caller.

len is the length of the string from 1 to 9.

ivalue is the output integer value.

irc is one of the following return codes:
irc=0 successful conversion.
irc=1 input string is incorrect.

DECOUT

This routine converts an integer value to a printable character string, in the same way as done by Fortran I format. The output is right justified in the area, preceded by blanks, and a minus sign ("-") is supplied if the number is negative. If the area is too short, it is filled with asterisks ("*"). This routine is re-entrant (it does not modify itself).

Calling Sequence: CALL DECOUT(string,len,ivalue)

Arguments:

string is the output character string.

len is the length of *string* set by the caller. The range is from 0 to 256.

ivalue is the input number.

DELAY

A call to this routine suspends the execution of the job for at least *n* seconds. Use the subroutine CLSOUT before a long delay if you wish to force all generated output to be printed before the delay.

Calling Sequence: CALL DELAY(n)

Arguments:

n is the number of seconds for the delay period. You should use a delay period of at least 1 second.

DSORT

DSORT is a generalized disk sort subroutine callable from many of MUSIC's high-level languages such as FORTRAN. For information see "Sorting Routines" in *Chapter 10. Utilities*.

ECHOIN

A call to this subroutine cancels the effect of a previous call to the NOECHO subroutine.

Calling Sequence: CALL ECHOIN

EOJ

This routine terminates the current job and sets the return code. (See also the GETRET subroutine.)

Calling Sequence: CALL EOJ(irc)

Arguments:

irc is an integer number supplied by the caller. It is put into register 15 when the job ends. A value of 0 usually means normal end, while a nonzero value usually indicates an error condition.

EQUAL

This subroutine is a FORTRAN LOGICAL function used for comparing bytes. If the compared bytes are identical, EQUAL is set to TRUE. If they are not identical, EQUAL is set to FALSE. This function must be declared as LOGICAL in your program. (See also EQUALB.)

Calling Sequence: v=EQUAL(a,b,n)

Arguments:

a is one location of bytes.

b is another location of bytes.

n is the number of bytes to compare. If *n* is less than 1, EQUAL is set to TRUE.

Example:

```
LOGICAL EQUAL
REAL *8 ALPHA,BETA
:
IF(EQUAL(ALPHA,BETA,6)) GO TO 10
```

EQUALB

This is a FORTRAN LOGICAL function used for comparing bytes. If the compared bytes are identical, EQUALB is set to TRUE. If they are not identical, EQUALB is set to FALSE. This function must be declared as LOGICAL in your program.

Calling Sequence: v=EQUALB(a,ka,b,kb,n)

Arguments:

a is one location of bytes.

ka indicates which byte to start with at location *a*. (The first byte is referred to as byte 1, not 0.)

b is another location of bytes.

kb indicates which byte to start with at location *b*. (The first byte is referred to as byte 1, not 0.)

n is the number of bytes to compare. If *n* is less than 1, EQUALB is set to TRUE.

Example:

```
LOGICAL EQUALB
REAL *8 ALPHA,BETA
:
IF(EQUALB(ALPHA,2,BETA,4,2)) GO TO 10
```

FBAS64

This subroutine decodes BASE64 data. This is the "base 64" encoding scheme used for e-mail mime, as defined in RFC 1521. Base 64 encodes any 8-bit text to printable characters, producing 4 output characters for each 3 input bytes. See the translate table in this routine for the 64 printable characters used. An additional character "=" is used for padding at the end of the encoded text if the original length is not a multiple of 3. See TBAS64 routine for additional notes on the pad character and for examples of encoding. (This routine is re-entrant.)

This routine decodes the base 64 data, back to the original data. See also: TBAS64 - encode to base 64; and UUDEC - uudecode-style decode.

Calling Sequence: CALL FBAS64(intxt,inlen,outtxt,outlen,retcod,displ)

Arguments:

intxt — input data (encoded as base 64). It is assumed to start on a quartet (group of 4 bytes in the encoded text) boundary. Note: intxt is destroyed by this routine (tranlated in place), so the caller should pass a copy of the original data, if it is to be used again. intxt is destroyed only up to (not including) the first blank or pad character ("=") or invalid character or to the end of the last full input quartet.

inlen — length of input data. If 0 or less, this routine sets outlen=0, retcod=0, and displ=0 and no other action is taken.

outtxt — area to receive output (decoded) data. It must be large enough to hold the output data. Maximum possible output is ((inlen+1)/4)*3 bytes, where / is integer divide i.e. discard the remainder.

outlen — this argument sets outlen to the length of the output data. This is usually a multiple of 3, but may be otherwise if the pad character ("=") is found in the input, indicating end of data. outlen is always the number of bytes stored into outtxt.

retcod,displ — the routine sets these integer arguments to indicate various cases, errors, and ending conditions. retcod is a return code and displ is a displacement within intxt, usually indicating how many input characters have been processed. Scanning stops when a blank is found, or after an EOF quartet (with "=" pad char) is found, or an invalid character is found; this we call the "scan end".

1. if the scan ends immediately follows a quartet, and the last quartet is not an EOF quartet, retcod=0 and displ=displacement to scan end i.e. 4*(number of quartets processed). outlen=3*(number of quartets processed).
2. if scan ends immediately follows an eof quartet, displ is set as in case (1), and retcod=-1 (if quartet is of the form "xxx=") or -2 (if "xx=="). an exception is that if the second "=" is not actually present in "xx==", displ is to after the first "=", i.e. 4*(no. of quartets) - 1. outlen=exact length of original text.
3. if scan end is within a quartet (other than the exception in case (2)), retcod=number of of chars in last quartet before the scan end i.e. 1 to 3, and displ=displacement to end of the last complete quartet. outlen=3*(number of complete quartets processed). only the first displ bytes of intxt are destroyed; the partial quartet at the end is intact, and can be concatenated with later data and passed to a subsequent call to this routine.
4. if an invalid character is found, retcod=4 and displ=displacement to the bad character. any preceding complete quartets are processed, and outlen=3*(the number of them).

Examples:

In these examples, x and y in outtxt are not the actual output characters, but indicate the form of the output.

```
intxt='abcdefgh',inlen=8: outtxt='xxxyyy',outlen=6,
       retcod=0,displ=8

intxt='abcd efgh',inlen=9: outtxt='xxx',outlen=3,
       retcod=0,displ=4

intxt=' ',inlen=0: outtxt='',outlen=0,retcod=0,displ=0
```

```
intxt='abcdefg=',inlen=8: outtxt='xxxyy',outlen=5,
       retcod=-1,displ=8

intxt='abcdef==',inlen=8: outtxt='xxxy',outlen=4,
       retcod=-2,displ=8

intxt='abcdef=',inlen=7: outtxt='xxxy',outlen=4,
       retcod=-2,displ=7 (this is the exception in case (2).)
    (same result for intxt='abcdef= ',inlen=8)

intxt='abcde',inlen=5: outtxt='xxx',outlen=3,retcod=1,displ=4

intxt='abcdefg',inlen=7: outtxt='xxx',outlen=3,
       retcod=3,displ=4

intxt='abcdef*h',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=6

intxt='abcde===',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=5

intxt='abcdef=*' or 'abcdef=h',inlen=8: outtxt='xxx',outlen=3,
       retcod=4,displ=6 (the "=" is the bad char because it is
       not followed by another "=")
```

Notes:

1. intxt argument is modified by this routine. See description of intxt above.

2. Other 3-to-4 encoding schemes may use a different translate table and pad character, and may handle lengths which are not a multiple of 3 differently.

3. This routine assumes the base 64 encoded text contains EBCDIC (not ASCII) printable characters. If the input is ASCII, the caller must convert the input to EBCDIC before calling this routine.

FILL

This routine sets each byte of an area to a specified character.

Calling Sequence: CALL FILL(area,len,char)

Arguments:

area is the name of the storage area.

len is the length in bytes of the storage area. If the *len* is 0 or negative, no bytes are changed.

char is the character to use to fill in the area.

Example:

```
CALL FILL(RECORD,80,' ')
```

FILMSG

Get descriptive error text corresponding to a file error condition. Refer to the section "Dynamic Access to Files" for a description of this routine.

FIXSCR

This subroutine translates the hex characters x'00' to x'3f' to x'40 (blank). Under certain conditions the hex characters between 00 x'00' and x'3f' combine with printable characters to accidentally produce 3270 data stream orders. These will disrupt the screen format when FSIO requests or panel are used. FIXSCR prevents these characters from damaging the screen format.

Calling Sequence: CALL FIXSCR(a,len)

Arguments:

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. no action if length is 0 or less.

FNDALL

This routine finds all the characters of a string that are one of the group of characters specified in *reference string*. All reference characters located will have their relative position in the string returned, as well as the number found.

Calling Sequence: CALL FNDALL(string,strlen,refstr,reflen,pos,numpos)

Arguments:

string is a character string of length strlen, that is to be verified.

strlen is the length of the string, in the range of 1 to 256.

refstr is the group of character(s) that constitute the set of characters that are to be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen is the length of the *refstr,* in the range 1 to 256.

pos is an integer array where the relative character positions of matched *refstr* characters are returned. *pos* should be dimensioned to the byte length of *string,* that is *pos(strlen).* If a character is detected in *string,* and is one of the reference characters, its relative position is returned in *pos(i).*

numpos is an integer where the number of relative character positions of matched *refstr* characters are returned.

Example:

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10

CALL FNDALL(STRING,STRLEN,REFSTR,REFLEN,POS,NUMPOS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10
POS    --> 3, 4, 6, 7, 0, 0, 0, 0
NUMPOS --> 4
```

FNDCHR

This routine finds the first occurrence of any of the characters specified by a *reference character string* in string. If a reference character is located, its relative position in the string is returned.

Calling Sequence: CALL FNDCHR (string,strlen,refstr,reflen,pos)

Arguments:

string	is a character string of length *strlen*, that is to be scanned.
strlen	is the length of the string, in the range of 1 to 256.
refstr	is the group of character(s) that constitute the range of characters, that is to be searched for, in *string*. This character string can be called the *reference string* and the characters *reference characters*.
reflen	is the length of the *refstr*, in the range of 1 to 256.
pos	is an integer returned that is set to 0 if none of the characters of string are also in the reference string, *refstr*. If a character is detected in *string*, that is one of the reference characters, its relative position is returned in *pos*.

Example:

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11

CALL FNDCHR(STRING,STRLEN,REFSTR,REFLEN,POS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11
POS    --> 3
```

FRMTDA

(Fortran G1 only) A call to this subroutine writes blank records throughout the direct access file defined on MUSIC I/O unit number *n.* It can be used to erase all previous contents of the file.

Calling Sequence: CALL FRMTDA(n)

FRSTOR

This subroutine frees main storage for use via an array in an os-mode program (normally FORTRAN). FRSTOR does a FREEMAIN to free a specified amount of storage starting at a specified offset from a given array. See also GTSTOR.

Calling Sequence: CALL FRSTOR(amt,a,offset,retcod)

Arguments:

amt	number of bytes to be freed. This number is always rounded up to a multiple of 8. *amt<=0* is a no-operation. *amt* is rounded up to a multiple of 8 by freemain.
a	an array relative to which the storage area is referenced.
offset	distance in bytes from the array to the storage area. The allocated area is always on a doubleword boundary. The area must be on a doubleword boundary.
retcod	return code: 0 successful. 1 requested storage not available. 2 invalid request or argument value (e.g. area to be freed is not on doubleword boundary). 3 not OS mode.

Notes:

1. An attempt to free storage that you did not get, or that has already been freed, can result in job termination by the GETMAIN/FREEMAIN processor in OSTRAP.

2. neither get nor free clears storage. The caller should do this (by calling the ZERO subroutine) if desired.

3. If free storage is fragmented, a request for maximum available storage returns the largest contiguous free area, which may be much less than the total free storage.

GETID

This routine gets the ownership id from a file name or a data set name.

Calling Sequence: CALL GETID(type,name,lname,idout,pos)

Arguments:

type — ID type:
- 1 name is a file name, and end of id is indicated by a colon (:), as in userid:name_proper
- 2 name is a UDS data set name, and end of ID is indicated by a period (.), as in USERID.ABC.DEF. also, if the DSNAME does not contain a period, and is length 4 or more, the userid is assumed to be the first 4 characters. Any other value for type sets IDOUT=' ',POS=1.

name — (input) the name to be processed.

lname — (input) max length of the name. The name is ended by a blank or after lname characters, whichever comes first.

idout — (output) 16-char userid from the name, padded with trailing blanks if necessary, or blanks if the name does not start with a 1 to 16 character userid.

pos — (output) integer position of the first character of the name proper, after the userid. Value 1 means no valid userid was present. Value *n* means the name proper starts at the n'th character of the name (counting from 1). n>1 means a valid userid was found.

GETOP

This subroutine gets the parameters from a character string. It is used by commands /LIST, /SAVE, /PURGE, /INPUT, etc. and by some utility programs.

Parameters in the string are separated by one or more blanks or commas, and may be preceded by leading blanks (but not leading commas - leading commas are considered to be part of the first parameter). Optionally, a command keyword at the beginning of the string may be skipped.

Calling Sequence: CALL GETOP(ctl,prmfld,prmlen,optbl,maxop,kwtbl,&err)

Arguments:

ctl is an integer containing option bits in the 4th byte:
- x'01' the input string (prmfld) starts with a command keyword which should be skipped. This is done by searching for the first blank and starting the parameter scan after that blank. If no blank is found, there are considered to be no parameters.

(other bits reserved)

prmfld is the input character string containing the parameters. It may have leading and trailing blanks (after the command keyword has been skipped, if x'01' option in *ctl).*

prmlen is the length (0 or more) of the character string.

optbl is the table to hold the parameter information. The first word will be set to the number of parameters

found (0 or more). The rest of the table consists of entries, each 3 words long.

1st word of entry: address of the parameter in the character string.

2nd word of entry: length of the parameter.

3rd word of entry: if the parameter is 1 to 9 decimal digits, this is the numeric value (0 or more). Otherwise the value is negative. The first byte is x'80', or x'80'+n if the parameter matches the keyword with id n in the keyword table *(kwtbl)*. Note: if a keyword in *kwtbl* is all decimal digits, The 3rd info word does not have the numeric value but starts with x'80'+n.

maxop is the maximum number of parameters that *optbl* can hold. the total size of the table is *1+3*maxop* words.

kwtbl is the table of special keywords which may occur as parameters. Each is given a non-zero id number n, which is returned in the *optbl* as described above. Different keywords may have the same id number. Each entry is of variable length:
1st byte: length of keyword, or 0 meaning end of table.
2nd byte: length of minimum abbreviation.
3rd byte: the id number n.
remaining bytes of entry: the keyword.

&err is the alternate return is taken if there are more than *maxop* parameters. The first *maxop* parameters are still put into the table, and the first word is set to *maxop*. For assembler callers, output r15=0 means no error, r15=4 means too many parameters.

GETRET

This routine returns the return code set by the previous job. This also includes jobs done via the NXTCMD subroutine.

Calling Sequence: CALL GETRET(irc)

Arguments:

irc is an integer number representing the return code set by the previous job.

GTSTOR

This subroutine gets main storage for use via an array in an OS-mode program (normally FORTRAN). GTSTOR does a GETMAIN to get a specified amount (or the max amount available) of main storage, and passes back an offset "T", which is the distance in bytes from the start of a given array "A" to the start of the allocated storage. A(T+1) then references the first element of the storage. The array is normally declared as size 1 in the calling program. See also FTSTOR.

Calling Sequence: CALL GTSTOR(amt,a,offset,lenget,retcod)

Arguments:

amt number of bytes to get. This number is always rounded up to a multiple of 8. *amt=0* requests the maximum storage available.

a — an array relative to which the storage area is referenced.

offset — distance in bytes from the array to the storage area. The allocated area is always on a doubleword boundary. This is set to 0 if there is an error.

lenget — the number of bytes to get(a multiple of 8). If there is an error, *lengot* is set to 0.

retcod — return code:

0 successful.
1 requested storage not available.
2 invalid request or argument value (e.g. amt<0 for get).
3 not OS mode.

Note: GTSTOR does not clear storage. The caller should do this (by calling the ZERO subroutine) if desired.

GULPDF

This subroutine is used to define a file to be processed using GULPRD and/or GULPWR subroutines for efficient reading and writing of large amounts of data.

Calling Sequence: CALL GULPDF(fileno {,sblk,filid} ,blksiz,numblk)

Arguments:

fileno — specifies the unit number and must be an integer from 1 to 15 inclusive. These numbers correspond to a /FILE statement defining a UDS-type disk data set or file.

blksiz — is the size of each block of data (in bytes) and is limited only by the size of the array to be used for data.

numblk — The program sets *numblk* to the number of blocks of data which can be stored in the data set.

sblk — is optional, it specifies the physical record number (number of the 512 byte block) to be used as the first block of a logical data set. If *sblk* is used then *filid* must be specified also.

Filid — is a logical unit number (specified by the user) by which this logical file is to be referenced, and must be a value from 1 to 32 inclusive: All arguments are integers.

Notes:

1. If *sblk* and *filid* are specified, the program can be called more than once for the same *fileno* permitting several logical files to be defined on one data set. If *sblk* and *filid* are not specified, they are assumed to be equal to 1 and fileno, respectively.

2. When the data set is originally allocated, a record size of 128 bytes should be specified. For a file, the record format should be F or U. If the file is to have n blocks of blksiz bytes each, the number of records specified at allocation time should be (with the result of the division truncated before multiplication):

$$4n \frac{(blksiz + 511)}{512}$$

3. The actual amount of disk space used for each logical block is a multiple of 512 bytes. For this reason, very small blksize specifications, or those just slightly larger than multiples of 512 tend to waste disk space.

4. No other I/O statements (direct access or sequential) should be used on data sets used by these routines.

5. The GULP routines can be used only with UDS or files on disk. They cannot be used with tape files.

6. When using GULPWR to write to a new file for the first time, *gaps* must not be left between blocks. For example, you may not write blocks 1,2, and 5 in that order, because blocks 3 and 4 would be a gap. A sequence such as 1,2,3,2,4,5 is allowed. Also, you may not read a block which has never been written. These restrictions do not apply to UDS files.

GULPRD

This subroutine is used to read a file defined by a call to the GULPDF subroutine.

Calling Sequence: CALL GULPRD(filid,blkno,array {,len})

or

CALL GULPRD(fileno,blkno,array {,len})

Arguments:

filid specifies the logical unit defined in the call to GULPDF. If it was not specified, *filid* should be the *fileno* specified in the call to GULPDF.

blkno is the number of the logical block at which reading is to begin.

array is the variable into which the data is to be read.

len is the number of bytes of data to be read, and its value must be from 1 to *blksiz*. *blksiz* is the argument supplied with GULPDF.) *Len* is optional and if omitted, *blksiz* is used.

Notes:

1. For a given block, if more data is read than had been written, the excess data has an unpredictable value.

2. All arguments are integers, except array which can be any type.

GULPWR

This subroutine is used to write a file defined by a call to GULPDF. (Arguments follow the same rules as GULPRD.)

Calling Sequence: CALL GULPWR(filid,blkno,array {,len})

or

CALL GULPWR(fileno,blkno,array {,len})

Arguments:

filid	specifies the logical unit defined in the call to GULPDF. If it was not specified, *filid* should be the *fileno* specified in the call to GULPDF.
blkno	is the number of the logical block at which writing is to begin.
array	is the variable into which the data is to be written.
len	is the number of bytes of data to write, and its value must be from 1 to *blksiz*. *blksiz* is the argument supplied with GULPDF.) *Len* is optional and if omitted, *blksiz* is used.

HWORD

This subroutine is a FORTRAN INTEGER function which returns the first two bytes of the argument *x*. Your program must declare this function as INTEGER.

Calling Sequence: n=HWORD(x)

ITSFxx

The following are indexed text search (ITS) retrieval subroutines for basic indexing:

ITSFID	initialization/dynamic call
ITSFIW	initialization/workarea call
ITSFOP	open a search set
ITSFSS	search string
ITSFSW	search word list
ITSFOR	order result list
ITSFRE	retrieve results
ITSFCL	close search

See the topic "ITS Subroutines" later in this chapter for more information.

I2C

This routine converts an integer to its EBCDIC character representation.

Calling Sequence: CALL I2C(i,string,sublen)

Arguments:

i	any 4 byte location to be converted to its EBCDIC character representation.
string	is the output string where the EBCDIC character representing the original string is returned. The character string representing the number is left justified in *string*. *String* must be at least 12 bytes long.
sublen	is a returned value that contains the length of the character string stored in string.

Example:

Given that:

```
I      --> 888

CALL I2C(I,STRING,SUBLEN)
```

Returns:

```
I      --> 888
STRING --> '888bbbbbbbbb'
SUBLEN --> 3
```

I2X

This routine converts an integer to its hexadecimal character representation.

Calling Sequence: CALL I2X (i,hexstr)

Arguments:

i	any 4 byte location to be converted to its hexadecimal representation.
hexstr	is the output string where EBCDIC hexdigits representing the original string are returned. *Hexstr* must be at least 8 bytes long.

Example:

Given that:

```
I      --> 249

CALL I2X(I,HEXSTR)
```

Returns:

```
I      --> 249
HEXSTR --> '000000F9'
```

KEEPIN

This subroutine is used to keep the contents of the input area for 3270-type workstations only. When a response to a conversational read is entered in the input area, it is not erased when the ENTER key is pressed. (See CLRIN.)

Calling Sequence: CALL KEEPIN

LAND

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *and* of two fullword operands *a* and *b.*

Calling Sequence: n=LAND(a,b)

Example:

```
N=LAND(ABLE,BAKER)
```

LBCOMP

This subroutine is a FORTRAN INTEGER function used for comparing bytes. The function returns a value of -1, 0, or 1 corresponding respectively to whether *a* is less than, equal to, or greater than *b.*

Calling Sequence: m=LBCOMP(a,ka,b,kb,n)

Arguments:

a is one location of bytes.

ka indicates which byte to start with at location *a.* (Bytes are numbered starting at 1, not 0.)

b is another location of bytes.

kb indicates which byte to start with at location *b.* (Bytes are numbered starting at 1, not 0.)

n is the number of bytes to compare. If *n* is less than one, the function returns a value of zero.

Example:

```
IF(LBCOMP(ALPHA,1,BETA,3,2)) 10,20,30
```

LBMOVE

This subroutine is used to copy bytes from one location in main storage to another location.

Calling Sequence: CALL LBMOVE(a,ka,b,kb,n)

Arguments:

a is the location in main storage of the bytes to be moved.

ka indicates which byte to start with at location *a.* (Bytes are numbered from 1, not 0.)

b is the location in main storage to move the bytes from location *a.*

kb indicates which byte to move to at location *b.* (Bytes are numbered from 1, not 0.)

n is the number of bytes to move. If *n* is less than one, no action is taken.

Example:

```
DIMENSION ABLE(20),BAKER(20)
:
CALL LBMOVE(ABLE(5),2,BAKER(2),3,40)
```

LCOMP

This subroutine is a FORTRAN INTEGER function is used to compare bytes. The function returns a value of -1, 0, or 1 corresponding respectively to whether *a* is less than, equal to, or greater than *b*.

Calling Sequence: m=LCOMP(a,b,n)

Arguments:

a is one location of bytes.

b is another location of bytes.

n is the number of bytes to compare. If *n* is less than one, the function returns a value of zero.

Example:

```
IF(LCOMP(I,J,2)) 10,20,30
```

LCOMPL

This subroutine is a FORTRAN INTEGER function which is used to transform the single full word argument to its 1's complement.

Calling Sequence: n=LCOMPL(a)

LJUST

This routine left justifies a substring, within a 1 to 256 character string, defined as the first non-blank character from the left and the last non-blank character.

Calling Sequence: CALL LJUST(string,strlen,sublen,pad)

Arguments:

string is a 1 to 256 character string to be left justified.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is taken by LJUST.

sublen is the length of the character string defined by the first non-blank character from the right.

pad all trailing blanks in the string are converted to the pad character.

Example:

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
PAD    --> 'b'

CALL LJUST(STRING,STRLEN,SUBLEN,PAD)
```

Returns:

```
STRING --> '12345bbb'
STRLEN --> 8
SUBLEN --> 5
```

LMOVE

This subroutine is used to copy bytes in main storage.

Calling Sequence: CALL LMOVE(a,b,n)

Arguments:

a is the start of the location in main storage of the bytes to move.

b is the start of the location to receive a copy of the bytes.

n is the number of bytes to move. If *n* is less than one, no action is taken.

LN

This subroutine gets the length of a character string, not counting trailing blanks.

Calling Sequence: n=LN(charvar)

or

n=LN(area,len)

Arguments:

charvar is a VS Fortran character variable or character array item. The declared length is found from the extra arglist info passed by VS Fortran.

area is any character string, of length "len".

n is the returned length of the string, not counting trailing blanks. n=LN(area,len) is equivalent to n=LOCATE(area,-len, ' ',-1), i.e. backscan for a nonblank.

Notes:

1. The first form, with only 1 argument, is valid only if the argument is a VS Fortran character item. For a non-char item, the length is unknown and n=0 is returned. No error message is issued.

2. If "len" is 0 or negative, n=0 is returned.

LOCATE

This subroutine searches for the substring within a string. If the string is located *n* is set to the starting position (i.e. byte number, counting from 1) of the substring which matches *str*. If the search is unsuccessful, *n* is set to zero.

LOCATE may be used as a subroutine or as an integer function.

Calling Sequence: CALL LOCATE(a,la,str,len,n)

or

n=LOCATE(a,la,str,len)

Arguments:

a location of the string to be searched.

la is the length of string *a* It can be specified as a negative or positive integer. If positive, the search for the substring starts at the beginning of *a* and proceeds forward. If *la* is negative, the search starts at the end of *a* and searches backwards. *la* should not be 0.

str is the substring to search for.

len is the length of *str* (in bytes) and must have an absolute value between 1 and 256 inclusive. It can be negative or positive. If *len* is negative, the search is for the first string **not** equal to *str*. Note that if *len* is negative, n=0 means that all substrings of *a* are equal to *str*.

LOCNAM

This subroutine changes an output file name to make it local to the current directory (if necessary).

Calling Sequence: CALL LOCNAM(filnam)

Arguments:

filnam is a 64-char file name, that will be used for output or append. This routine adds ".\" (restrict the name to the current directory - or lower) to the front of the name if the name does not already start with any of the following:

id:	(a specific userid implies root of that id)
\	(root directory or a specific subdirectory)
.\	(current directory)
..\	(next higher directory)
/	(special names like /input imply the root)

This is similar to what the /PURGE command does.

For example, suppose an application lets a user specify a file to which some data is to be appended (e.g. mail copy function with the append option). The application should call this routine to adjust the name. If the user specifies file name "abc", this routine would change it to ".\abc". a name like "sam:abc" or "\mydir\abc" would be left as is.

If this routine is not called, there is real danger of a privileged user appending to a public file that he/she does not own, thus corrupting the public file without realizing it.

LOR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *or* of the two full word arguments *a* and *b.*

Calling Sequence: n=LOR(a,b)

LSHFTL

This subroutine is a FORTRAN INTEGER function which is used to perform a logical shift left, *k* bits, in the full word argument *a.* If *k* is less than 0, 0 is used. If *k* is greater than 32, 32 is used.

Calling Sequence: n=LSHFTL(a,k)

LSHFTR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical shift right, *k* bits, in the full word argument *a.* If *k* is less than 0, 0 is used. If *k* is greater than 32, 32 is used.

Calling Sequence: n=LSHFTR(a,k)

LXOR

This subroutine is a FORTRAN INTEGER function which is used to perform a logical *exclusive or* of the two full word arguments *a* and *b.*

Calling Sequence: n=LXOR(a,b)

MA2E

This subroutine translates 8-bit ASCII to (MUSIC) EBCDIC. See also: ME2A, A2E, and E2A subroutines.

Calling Sequence: CALL MA2E(a,len)

Arguments:

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. No action is taken if length is 0 or less. (This routine is re-entrant.)

Notes:

1. This is a one-to-one mapping between the 256 ASCII and 256 EBCDIC characters. When routines MA2E and ME2A are used successively on the same data, there is no net change. It can therefore be used for storing extended ASCII (accented, foreign, and graphics chars, etc.) or binary ASCII on MUSIC in EBCDIC. When the data is later translated back to ASCII, nothing will be lost.

2. Most ASCII control characters are translated correctly. In particular: cr, lf, ht (tab).

3. This translation is very close to that done by ind$file (3270 file transfer). ME2A's translation of EBCDIC 6a to d5 is one exception; ind$file translates it to 7c, making ind$file not one-to-one.

4. Some notable characters:

```
EBCDIC          <--->      ASCII
 4a (cent sign)              e5
 4f (solid vert bar)         7c (split vert bar)
 5f (not sign)               5e (caret)
 6a (split vert bar)         d5
 79                          60 (grave accent)
```

MCNCAT

This routine can replace VS Fortran's CNCAT# (in IFYCNCAT or AFBCNCAT) concatenation routine, in cases where the Fortran run-time library routines are not available (e.g. MAIL).

Place /INC $SUB:MCNCAT.OBJ in the program or LKED the file. (Note: This routine should *not* be added to the subroutine library.)

Calling Sequence: CALL MCNCAT(src1,len1,src2,len2,...,dest,destln)

Entry point CNCAT# is equivalent to MCNCAT. This is equivalent to: dest=src1//src2//...

Strings can be any length, including 0. If *len<=0*, string is considered to be null. The resulting string is truncated or blank-padded to match the target string length. Immediate return if # arguments is not even and >=2.

ME2A

This subroutine translates (MUSIC) EBCDIC to 8-bit ASCII. See also: MA2E, A2E, and E2A subroutines. (This routine is re-entrant.)

Calling Sequence: CALL ME2A(a,len)

Arguments:

a is the data to be translated.

len is the number of bytes to be translated (0 or more). length may exceed 256. No action if length is 0 or less.

Notes:

1. This is a one-to-one mapping between the 256 ASCII and 256 EBCDIC characters. When routines MA2E and ME2A are used successively on the same data, there is no net change. It can therefore be used for storing extended ASCII (accented, foreign, and graphics chars, etc.) or binary ASCII on music in EBCDIC. When the data is later translated back to ASCII, nothing will be lost.

2. Most ASCII control characters are translated correctly. in particular: cr, lf, ht (tab).

3. this translation is very close to that done by ind$file (3270 file transfer). me2a's translation of EBCDIC 6a to d5 is one exception; ind$file translates it to 7c, making ind$file not one-to-one.

4. Some notable characters:

```
EBCDIC          <--->      ASCII
 4a (cent sign)              e5
 4f (solid vert bar)         7c (split vert bar)
 5f (not sign)               5e (caret)
 6a (split vert bar)         d5
 79 (grave accent)           60 (grave accent)
```

MODFLD

This routine is used to re-define the attributes of modifiable fields by field number of screens designed using PANEL. The attributes that can be modified are: protection, intensity, hide, and skip. See the PANEL documentation in *Chapter 10. Utilities* for details about these.

Calling Sequence: CALL MODFLD(panel,ifld,irc,{iprt,intens,ihide,iskp,fldlen})

Arguments:

panel is the panel name whose attributes are to be modified. Note that *panel* must be declared as external in the calling program.

ifld is the field number whose attributes are to be modified.

irc is the return code.
- =0 normal return, instructions followed as specified.
- =1 invalid field number, *ifld* is either less than zero or greater than the highest field number available for this panel.
- =2 skip setting could not be set because the field following the target field did not have an attribute of x'6c' or x'7c' (protected, nondisplay).

iprt is the protection flag:
- <=-1 set to unprotected
- =0 make no change to current field protection status
- >=1 set to protected

intens is the intensity flag:
<=-1 set to low intensity
=0 make no change to current field intensity status
>=1 set to high intensity

ihide is the hide flag:
<=0 make no change to current hide setting.
>=1 set attribute to hide (nondisplay, noprint, nondetectable)
To undo hide, reset the intensity; hide is a form of intensity.

iskp is the skip flag:
<=-1 set to no skip
=0 make no change to current skip status
>=1 set skip
Skip for a field can only be set if the field is immediately followed by a protected, numeric, nondisplay/ nondetectable/noprint field.

fldlen the length of the panel field is returned.

Note: *iprt*, *intens*, *ihide*, and *iskp* are optional positional arguments. If you do not wish to choose one of these options, assign a value of 0 (zero). If the option(s) appears at the end of the sequence, it may be left out.

Examples:

1. The following shows how to change the intensity of field number 2.

```
    CALL MODFLD(PANEL1,2,IRC,0,1,0,0)
or CALL MODFLD(PANEL1,2,IRC,0,1)
```

2. The following example indicates that field number 3 is to be hidden.

```
    CALL MODFLD(PANEL1,3,IRC,0,0,1)
```

MODOPT

This routine is used to modify PANEL options previously defined using the PANEL subsystem. The options that can be modified are:

1 upper/lower case translation.
2 PF13-PF24 translated to PF1-PF12.
3 PA2 automatic reshow.
4 PF12 as print screen.
5 help screen availability.
6 the help screen to be displayed (by name).

Calling Sequence: CALL MODOPT(panel,irc,icode,iset)

Arguments:

panel is the panel name whose attributes are to be modified. Note that PANEL must be declared as external in the calling program.

irc is the return code.
=0 normal return, instructions followed as specified.
=1 invalid icode number, ICODE <=0 or > 5.

icode is option number to be reset.
=1 upper/lower case translation.
=2 PF13-PF24 translated to PF1-PF12.
=3 PA2 automatic reshow.
=4 PF12 as print screen.
=5 help screen available.
=6 help screen to be displayed (by name)

iset is the option setting desired:
<=-1 turn off option.
=0 make no change to current setting.
>=1 turn on the option.
= help panel name declared external in the calling program.

Examples:

```
CALL MODOPT(PANEL1,IRC,6,HELP1)

CALL MODOPT(PANEL1,IRC,1,-1)
```

MOVE

This subroutine copies full words from one location to another. *n* full words starting from location *a* to locations starting at location *b*.

Calling Sequence: CALL MOVE(a,b,n {,m})

Arguments:

a is the location of the words to be moved.

b indicates the location to move the words.

n is the number of full words. If *n* is less than 1, no action is taken.

m is optional, it specifies the length of each element. If *m* is omitted, 4 is assumed.

NOCRLF

This subroutine is for ASCII terminals only. A call to this subroutine suppresses the carriage return-line feed which normally occurs at the end of each line of output. (See CRLF.)

Calling Sequence: CALL NOCRLF

NOECHO

This subroutine is only used for 3270-type workstations. A call to this subroutine causes the contents entered to the input area of the screen, when responding to a conversational read, not to be displayed on the output area of the screen when the ENTER key is pressed. (See ECHOIN.)

Calling Sequence: CALL NOECHO

NONCAN

A call to this subroutine makes the program non-cancellable. That is, the /CANCEL command can not be used to terminate the program. Use CALL CANCAN to remove the effect of CALL NONCAN. Note that a call to NONCAN only takes effect after the next conversational read or call to CLSOUT.

Calling Sequence: CALL NONCAN

NOPAUS

A call to this subroutine inhibits the system function of stopping every few lines to give the user a chance to cancel the job that is running. (It is meaningful only with 1050 terminals.) See also PAUSE.

Calling Sequence: CALL NOPAUS

NOSHOW

This subroutine is only used for 3270-type workstations. A call to this subroutine causes the characters being entered to the input area of the screen, when responding to a conversational read, not to be displayed. (See SHOWIN.)

Calling Sequence: CALL NOSHOW

NOTRIN

This subroutine suppresses the translation of terminal input data from the terminal code to EBCDIC. This can be used for special applications on ASCII terminals, where the program requires the original terminal data stream. The translation of terminal output data can be accomplished using the X'41' carriage control. Normal translation can be resumed by calling the subroutine TRIN.

Calling Sequence: CALL NOTRIN

NPRMPT

A call to this subroutine can be used to delete the conversational read prompt message entirely. If however, the workstation is set up to pause periodically during output (for example, the /PAUSE command has been issued from a workstation), a question mark (?) is issued as a prompt. (See PROMPT.)

Calling Sequence: CALL NPRMPT

NSGNOF

A call to this subroutine removes the request to schedule a /OFF next. It can be used to negate the effect of a previous call to SIGNOF.

Calling Sequence: CALL NSGNOF

NXTCMD

A call to this subroutine causes the specified command, or program, to be executed. Depending on the options specified, the command is either run concurrently with the calling program using multi-tasking support or is run after the calling program terminates using program chaining.

Calling Sequence: CALL NXTCMD(cmd,len{,opt})

Arguments:

cmd is the command string to be executed. If the command string specified is not a valid MUSIC command an attempt is made to schedule the user program with that name. The command or program name part of the string should be in upper case. The second part of the command string is the parameter (PARM field).

len is the total length of the command string and should not exceed 196. If the string is too long, the excess characters at the end are ignored.

opt is an optional bit-string value that controls the execution of the command. Only the bits the low order byte are used. The high order bytes must be zero. The X'80' bit determines if the request is for multi-tasking execution or program chaining.

Program Chaining

In program chaining the system remembers the command specified in the NXTCMD call and executes it automatically when the calling program terminates. The following options are available.

x'80' Must be zero.

x'40' Not used.

x'20' Not used.

x'10' Not used.

x'08' Not used.

x'04' Not used.

x'02' Not used

x'01' Non-cancelable.

Commands that are processed directly by MUSIC's workstation command scanner are not supported by program chaining. These are /COMPRESS, /CTL, /DISCON, /EXEC, /NS, /PAUSE, /PROMPT, /REQUEST, /RUN, /STATUS, /TIME, /TEXT, /TABIN, /TABOUT, /USERS, /WINDOW.

Multi-Tasking

When the multi-tasking option is used, the system creates a new session and runs the command immediately. This is referred to as the child task. The original session is called the parent. There are a number of options available that define the relationship of the child and the parent. Since multi-tasking is based on the systems multi-session support, multi-session commands such as /NEXT and /PREVIOUS can be used to switch between child and parent.

x'80' Must be set to 1.

x'40' Parent will wait for child task to end. This option is ignored if the parent or child is a background task (BTRM).

x'20' Delete child task at child end of job.

x'10' Displays job startup messages

x'08' Control of the workstation remains with the parent task once the child is started.

x'04' Hide parent task. Multi-session commands cannot be used to return to parent before child terminates.

x'02' Run task in background (BTRM).

x'01' Make child task non-cancelable.

x'0C' Delete child when parent task terminates.

Examples:

```
CALL NXTCMD('HELP SORT',9,128+64+32)
```

This invokes the HELP facility for topic "sort". The parent task waits, and the new task (help) is deleted when the help program ends.

```
CALL NXTCMD('EDIT FILE1',10)
```

This uses program chaining to call the editor.

NXTPGM

A call to this subroutine causes the specified program to be executed when the current job terminates. This routine can also be used to schedule an *always* program. An *always* program is one that is run whenever the workstation would otherwise return to command mode and can be used to run command mode replacement programs such as TODO, or provide sophisticated program chaining facilities. A call to this subroutine with a blank name cancels the previous call.

Calling Sequence: CALL NXTPGM('name '{,'parm',len,opt})

Arguments:

name is the name of the file which contains the job to be executed. The length of *name* can be from 1 to 22 characters. If name is less than 22 characters in length, it must be terminated with a blank. If the option argument is 16 or 17, the file name can be up to 64 characters long and must be terminated with a blank.

parm is a character string which will be passed to the program that is to be executed. The maximum length of parm is 74 characters.

len is the length of the parameter string. If it is not specified, the parameter string must be terminated by a $ sign (which is not part of the actual parm).

opt is one of the following numbers informing the system that the program is:

1 - non-cancelable
4 - an "always" program
5 - both, non-cancelable and always program
16 - long file name (up to 64 characters)
17 - non-cancelable and long file name

Note: If you do not wish to pass any *parm* to the program, either specify only the name argument, or specify the 2nd argument as 0. The *parm* field is not used by *always* programs.

Examples:

```
CALL NXTPGM('name ')

CALL NXTPGM('name ','parm$')

CALL NXTPGM('name ','parm',len)

CALL NXTPGM('name ','parm',len,1)     (non-cancelable)

CALL NXTPGM('name ',0,0,4)            (always program)
```

NXWORD

This subroutine gets the next word from a string. **Warning:** This routine makes assumptions about how the TOUC subroutine is coded.

Usage:

```
POS=1
DO UNTIL LENOUT=0
  CALL NXWORD(pos,a,lena,out,maxout,lenout,opt)
  ...(PROCESS THE WORD IN "OUT")...
END-UNTIL
```

Arguments:

pos indicates the starting position in the string. It should be set to 1 before calling NXWORD, to start at the beginning of the string (i.e. to get the first word). NXWORD sets *pos* to the position of the delimiter (blank or comma) following the end of the returned word, or to *lena+1* if no word found.

a is the string.

lena is the length of the string.

out is an area, of length *maxout*, to receive the word. The word is truncated or blank-filled to match the area. The search starts at position *pos* in the string. Delimiters (blanks and (optionally) commas) are skipped to get to the start of the next word. The word ends at the next delimiter or end-of-string.

maxout is the length of the output area.

lenout is the length of the word. If the output area is too small, the word is truncated to length *maxout*, but *lenout* is set to the true (longer) length. If there are no more words in the string, *lenout* is set to 0 and *out* is set to blanks.

opt is an (optional argument) numerical option value. option bits in the 4th byte are:

```
   x'01'  Treat commas as a delimiter, in addition to blanks.
   x'02'  Convert the word to upper case.
   x'04'  Consider all characters between ( ) as part of the
  keyword.
         e.g.  abc(xyz ttt )
               abc( 1,2)
               abc(xyz(1 2) fred)
```

If omitted, 0 is assumed.

Examples:

```
CHARACTER A*10/'ABCD EFG  '/,OUT*8
POS=1
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='ABCD    ', LENOUT=4, POS=5
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='EFG     ', LENOUT=3, POS=9
CALL NXWORD(POS,A,10,OUT,8,LENOUT)
  ... SETS OUT='        ', LENOUT=0, POS=11
```

See also: PROCOP, ABBREV, GETOP, and SEP.

OPNFIL

Dynamically open a file. Refer to the section "Dynamic Access to Files" for a description of this routine.

PANFLD

This routine queries infomation about PANEL fields.

Calling Sequence: CALL PANFLD(PANEL,RC,FLD,ROW,COL,FLDLEN,PROT,
INTENS,HIDE,SKIP,ATTR,TOTLEN)

Arguments:

PANEL — is the panel name whose info is to be queried. Note that PANEL must be declared as external in the calling program.

RC — is the return code.
- = 0 normal return, instructions followed as specified.
- = 1 invalid field number, ifield <0 or > than highest field number available for this panel.
- = 3 wrong number of arguements. the first 3 are required but no more than 12 arguments can be present.

The following are only given if FLD=0

- = 4 No field on this row, no parameters updated
- = 5 After the last field of a row, no parameters updated
- = 6 before the first field of a row information returned is about the first field
- = 7 in between fields of a row information returned is about the next field

FLD — is the field number whose info is to be queried. If FLD is 0, then ROW and COL are used to query the field and can be anywhere in the field.

ROW — is the row on the screen of the field.

COL — is starting column of the field on output, or is any column in the field on input if FLD=0.

FLDLEN — is the length of the panel field.

PROT — is the protection flag:
- = 0 Field is unprotected
- = 1 field is protected

INTENS — is the intensity flag:
- = 0 field is low intensity
- = 1 field is high intensity

HIDE — is the hide flag:
- = 0 field is not hidden
- = 1 field is hidden

SKIP — is the skip flag:
- = 0 Field is not skipped
- = 1 field is skiped

ATTR — is the attribute byte of the field.

TOTLEN — is the offset of this field from the beginning of the data portion of the COMMON block.

PARM

This subroutine is used to retrieve the parameter string which was passed to the calling program by a /PARM statement or by a parameter string specified on the /EXEC command.

Calling Sequence: CALL PARM(buf,buflen,prmlen)

Arguments:

buf	is the receiving area which is blanked out prior to receiving the parameter string. Leading and trailing blanks of the parameter string are not transferred to *buf*.
buflen	is the length of the receiving area which can have a maximum length of 256 characters.
prmlen	is set (by the routine) to the actual length of the parameter string that is moved to *buf*. If *prmlen* is longer than *buflen* then only the leftmost "buflen bytes of the parameter string are moved to *buf*.

PAUSE

A call to this subroutine cancels the effect of a call to NOPAUS.

Calling Sequence: CALL PAUSE

PROCOP

This subroutine processes an option item in the format abc(xyz).

The following option forms are accepted:

```
abc
abc(xyz)
abc=xyz
abc( xyz )
abc= xyz
```

Calling Sequence: CALL PROCOP(a,len,kwlen,subpos,sublen,numval)

Arguments:

a	is the option string. It should not have leading or trailing blanks.
len	is the length of a.
kwlen	is the (output) length of keyword part (abc), i.e. length up to first (or = or end of string.
subpos	is the (output) position in a of start of suboption (xyz), or 0 if no suboption.
sublen	is the (output) length of suboption (xyz), or 0 if none.
numval	is the (output) numeric value of mmm (if mmm is 1 to 8 digit chars), otherwise -1. mmm is xyz (ignoring leading and trailing blanks) if there is a suboption, otherwise mmm is abc.

Examples:

```
CALL PROCOP('123',3,N1,N2,N3,N4) --> 3,0,0,123
CALL PROCOP('12X',3,N1,N2,N3,N4) --> 3,0,0,-1
CALL PROCOP('AB(257)',7,N1,N2,N3,N4) --> 2,4,3,257
CALL PROCOP('AB=257',6,N1,N2,N3,N4) --> 2,4,3,257
CALL PROCOP('AB( 257 )',9,N1,N2,N3,N4) --> 2,5,3,257
CALL PROCOP('AB= 257',7,N1,N2,N3,N4) --> 2,5,3,257
CALL PROCOP('AB( 25X )',9,N1,N2,N3,N4) --> 2,5,3,-1
CALL PROCOP('AB(-257)',8,N1,N2,N3,N4) --> 2,4,4,-1
```

See also: NXWORD, ABBREV.

PROMPT

Cancels the effect of the NPRMPT subroutine.

Calling Sequence: CALL PROMPT

PURGE

This routine deletes a file from the Save Library.

Calling Sequence: CALL PURGE(filename,irc)

or

CALL PURGE(-1,longname,irc)

Arguments:

filname — is a file name, with or without the userid prefix. If the name is less than 22 characters long, it must be followed by at least 1 blank. The maximum length is 22 characters.

-1,longname — is a long file name, up to 64 characters. If the name is not the maximum length, it must be followed by a blank.

irc — is an integer return code, 0 meaning the file was deleted successfully. For the meaning of nonzero return codes (file not deleted), see the topic "Dynamic Access to Files" of this guide. Also, HELP is provided with the topic ERRORS. (For example, 30 means file not found, 33 means file in use.)

Note: This routine does not preserve R0, R1.

Example:

```
CALL PURGE('MYFILE ',K)
```

QFOPEN

Opens a file and defines a buffer area. Refer to the section "Dynamic Access to Files" for a description of this routine and QFCLOS, QFREAD, QFRBA, QFREW, and QFBKRD.

REREAD

(FORTRAN (G1) only) - allows rereading records as many times as desired, without the expense of physical I/O. The record can be read with the same or different formats and lists. REREAD must be called only once at the beginning of the FORTRAN program. Thereafter a formatted or list-directed read from unit 99 rereads the record read by the last formatted read. No other I/O statement may appear between the two READ statements.

The FORMAT and list should specify single record input, or the results are undefined. The following is an example of a case that should be avoided:

```
          READ(99,200)I,J
      200 FORMAT(I4)
```

Calling Sequence: CALL REREAD

Example:

```
*Go
list sample
*In progress
      CALL REREAD
    8 READ(5,1,END=10)I
    1 FORMAT(I1)
      GO TO (2,3),I
    2 READ(99,4)J
    4 FORMAT(1X,I10)
      WRITE(6,5)J
    5 FORMAT('0FORMAT 4 WAS USED, J=',I10)
      GO TO 8
    3 READ(99,6)A
    6 FORMAT(1X,F10.2)
      WRITE(6,7)A
    7 FORMAT('0FORMAT 6 WAS USED, A=',F10.2)
      GO TO 8
   10 STOP
      END
/DATA
1         4
2         4
*End
*Go
sample
*In progress
MAIN   = 000250
003668 BYTES USED
EXECUTION BEGINS

FORMAT 4 WAS USED, J=            4

FORMAT 6 WAS USED, A=         0.04
STOP        0
*End
*Go
```

RJUST

This routine right justifies a substring, within a 1 to 256 character string, defined as, the first non-blank character from the left and the last non-blank character.

Calling Sequence: CALL RJUST(string,strlen,sublen,start,pad)

Arguments:

string is a 1 to 256 character string to be right justified.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is taken by RJUST.

sublen is the length of the character string defined by the first non-blank from the left and the first non-blank character from the right.

start this is the starting byte of the substring justified on string.

pad all leading blanks in the string are converted to the pad character, after the text is justified.

Example:

Given that:

```
STRING --> 'bb12345b'
STRLEN --> 8
PAD    --> 'b'

CALL RJUST(STRING,STRLEN,SUBLEN,START,PAD)
```

Returns:

```
STRING --> 'bbb12345'
STRLEN --> 8
SUBLEN --> 5
START  --> 4
```

RSTART

A call to this subroutine initializes a general purpose random number generator routine. The basic technique used is a combination of a multiplicative congruential generator and a shift-register generator. Each time a random number is requested, two new numbers are generated and then combined to form the next uniformly distributed number of the sequence (or the number from which a normal or exponentially distributed number is derived).

Calling Sequence: CALL RSTART(i,j)

Arguments:

i is an integer which becomes the starter of the multiplicative congruential generator sequence. If *i* is zero then the sequence of numbers that it starts is zero, so that CALL RSTART (0,j) provides a pure shift register generator.

j is an integer which becomes the starter of the shift-generator. If *j* is zero then the sequence of numbers that it starts is zero, so that CALL RSTART(i,0) provides a pure multiplicative generator.

After calling RSTART, calls to functions UNI, VNI, RNOR, REXP, IUNI, and IVNI can be made to get the next random number in the sequence.

u=UNI(0) provides a normalized floating point variate uniformly distributed on (0<=u<1).

v=VNI(0) provides a normalized floating point variate uniformly distributed on (-1<=v<1).

r=RNOR(0) provides a normalized floating point variate from the standard normal distribution, with a mean of 0.0 and a variance of 1.0.

a=REXP(0) provides a normalized floating point variate with the standard exponential density e**(-y) where (y>=0).

i=IUNI(0) provides an integer variate uniformly distributed in the range (0<=i<2**31).

i=IVNI(0) provides an integer variate uniformly distributed in the range (-2**31<=i<2**31).

Example:

```
C  DISPLAY 50 RANDOM NUMBERS UNIFORMLY DISTRIBUTED BETWEEN 0 AND 1
      REAL X,UNI
      CALL RSTART(12345,98765)
      DO 10 N=1,50
      X=UNI(0)
10    WRITE(6,20) X
20    FORMAT(1X,F10.5)
      STOP
      END
```

Notes:

1. If RSTART(0,0) is used, every number generated will be zero. If CALL RSTART is not used, built-in starting values are used.

2. Because the basic random number is a combination of multiplicative and shift-register generators, the resulting sequence has a very large period (approximately 5 x 10**18). The random variables returned from the RNOR and REXP routines have exactly the required distribution, normal and exponential respectively. Both the RNOR and the REXP procedures are extremely fast, generating a variate in less than twice the time required to generate the uniform variate.

RVRS

This routine reverses a 1 to 256 character string by swapping the characters end to end.

Calling Sequence: CALL RVRS(string,strlen)

Arguments:

string is a 1 to 256 character string to be reversed.

strlen is the length of the string, an integer in the range of 1 to 256. When strlen =0 no action is

taken by RVRS.

Example:

Given that:

```
   STRING --> 'abcdefgh'
   STRLEN --> 8

CALL RVRS(STRING,STRLEN)
```

Returns:

```
   STRING --> 'hgfedcba'
   STRLEN --> 8
```

SAVREQ

A call to this subroutine schedules a /SV name command to be scheduled for execution. The command is executed automatically as soon as the program terminates execution, whether it terminates normally or abnormally. The save is not be executed if the job is canceled by the system because of excessive output, or if a call to SIGNOF is made after the call to SAVREQ, or if the job is canceled by a /CANCEL command.

Calling Sequence: CALL SAVREQ('name ')

Arguments:

name is the file name to be used and must be six characters in length.

SEP

This subroutine separates out fields separated by 1 or more blanks/commas.

Calling Sequence: CALL SEP(a,alen,maxnum,num,pos(i),len(i))

Arguments:

a	is the string to be processed (input).
alen	is the length of the string (input).
maxnum	is the size of arrays *pos* and *len* (input).
num	is the number of fields found (0 to *maxnum)* (output).
pos(i)	is the position of the i'th field (output). (Position 1 is the first character of the string.)
len(i)	is the length of the i'th field (output).

Notes:

1. If there are more than *maxnum* fields, the extra ones are ignored.

2. If the string contains only blanks or commas, or if *alen=0, num* is set to 0.

SETINF

Specify information about a new file to be opened by OPNFIL. Refer to the section "Dynamic Access to Files" later for a description of this routine.

SHOWIN

A call to this subroutine cancels the effect of a previous call to the NOSHOW subroutine.

Calling Sequence: CALL SHOWIN

SIGNOF

A call to this subroutine schedules a /OFF command which is executed automatically as soon as the program terminates execution, whether it terminates normally or abnormally, unless a call to SAVREQ is made after the call to SIGNOF. (See NSGNOF.)

Calling Sequence: CALL SIGNOF

SRTMUS

Generalized disk sort subroutine. For information see "Sorting Routines" in *Chapter 10. Utilities*.

SSORT

This subroutine is an efficient means of sorting an array (or any table of contiguous, equal-length entries) into ascending order based on a control field within each entry. Comparison is character-type rather than numeric.

Calling Sequence: CALL SSORT(array,num,len,keylen,{keypos})

Arguments:

array	is the table of entries to be sorted. If *array* is a logical*1 2-dimensional array, the dimensions of it would be (len,num). If *array* is a fullword integer array, the dimensions of it would be (len/4,num).
num	is the number of entries in *array*. If *num* is less than 2, the subroutine returns since no sort is required.

len is the length in bytes of each entry and must be a number from 1 to 256 inclusive.

keylen is the length in bytes if the control field to be used for the sort.

keypos is the starting position (byte number, counting from 1) of the control field within each entry. If *keypos* is omitted, 1 is assumed.

STATUS

This subroutine returns a formatted line consisting of time of day, current date, service units used, and number of users currently signed.

Calling Sequence: CALL STATUS(stline,work)

Arguments:

stline a 79 byte character string returned.

work a real*8 array of dimension 4 (real*8 work(4)).

Note: This routine is re-entrant.

STOPSK

A call to this subroutine terminates the effect of a /SKIP nnn command entered by the user. That is, lines written to the workstation after the call to STOPSK display regardless of any previous /SKIP nnn command. The normal effect of /SKIP nnn is restored after the first such line begins to display.

Calling Sequence: CALL STOPSK

SYSINE

A call to this subroutine indicates the status of the current job stream input file (default unit 5 from FORTRAN).

Calling Sequence: CALL SYSINE(mcod)

Arguments:

mcod is set by this subroutine to a value depending on current status:

0 No error conditions were associated with the last read operation from the job stream file.
1 End-of-file was encountered at the line read by last read operation.
2 Probable programmer error: for example, /INCLUDE nesting level exceeds 5, or an improper /INCLUDE command was found on the last read.
3 Loss of file integrity caused by an I/O error or some other problem. This could be caused by trying to read a load module file which was created with record format FC (the default). Load modules should be created with record format F. The MUSIC Systems Support Group should be advised if this error persists.
4 (or higher) File is not accessible. The file does not exist or is defined as private or execute only.

A user program is limited to 10 occurrences of error code 4 before the user's job is automatically terminated.

SYSINL

This subroutine can be used to find information about the current job stream input file (unit 5 from FORTRAN).

Calling Sequence: CALL SYSINL(fn,ln,nest,mcod,k,lln,maxdep)

Arguments:

fn is the file name currently being read, returned as 8 characters with the last two always blank.

ln is returned as the line number within the current file of the last line read, with 1 being the first line of a file (line number 0 may occur if the first line has not been successfully read).

nest is returned as the current /INCLUDE nest level, if the user sets nest to zero before calling the subroutine. If the user sets nest to a value between 1 and the current nesting level before calling the subroutine, then information about the specified nesting level is returned. (If nest is outside this range, the results of the call are undefined).

mcod returns the same result as a call to SYSINE.

k returns the flags at the current nest level, as defined for subroutine SYSINR. A value of 256 is added to k if the file is owned by the person running the program.

lln specifies a limiting line number, and if specified as -1, then no limit is set.

maxdep returns the maximum nest level the system can handle.

Note: File names beginning with a slash (/) character can be returned in the *fn* argument. The name /INPUT refers to the /INPUT file, /TRANX refers to the alternate input file used with /EXEC and /TRMIN means the conversational read spool file.

SYSINM

A call to this subroutine causes a message to be printed on I/O unit 6, of the form `**AT LINE nnnn of FILE xxxxxx` where *nnnn* is the current line number of the current input file *xxxxxx*. (This subroutine cannot be called from COBOL or VS Fortran programs as it is not compatible with the OS/MUSIC interface.)

Calling Sequence: CALL SYSINM

SYSINR

This subroutine allows the user to dynamically control the system input file (default unit 5 from FORTRAN).

Calling Sequence: CALL SYSINR(fn,ln,lln,nc,k)

Arguments:

fn is an eight character file name specified by the user, with the last two characters blank. The special name of '/TRMIN ' is used to request spooled conversational reads. For more information see *Chapter 4. File System and I/O Interface* of this guide.

ln specifies the line number to be read by the next operation.

lln specifies the limiting line number at which an end-of-file condition is to be reached. A value of -1 specifies no limit.

nc is a control argument. If *nc=1,* the nest depth is advanced by one and the named input file is set to be read at line number *ln* (with *ln=1* specifying the first line of the file). When the specified file is completely read (or up to line *lln)* the next read operation reads from the following line in the next higher level; i.e the next line in the file being processed at the time of this subroutine call. If the last read operation resulted in an end-of-file condition (including error conditions) then *nc=1* is assumed to be *nc=0.* You can make successive calls to SYSINR if you wish to set up an input file nest, and then your first read statement reads as specified by the last call. Successive calls, cannot in any case, exceed the maximum nesting depth of 5.

If nc=0 is specified, reading of the current file is terminated and the next read operation reads from line *ln* of file *fn.* If the file name *fn* is specified as numeric 0 then the file name is not changed from its current value. This function could be used to skip lines in the current file, although it is more efficient to skip lines by actually reading them.

k is a variable which specifies control flags allowing specification of the same options allowed on the /INCLUDE command. These options are coded in numbers, and the sum of these option numbers form the contents of variable *k* completely replacing those options currently in effect at the specified level.

Option	Option Number
NEST	1
EOF	2 (an end-of-file return is always taken when there is no input left to be read)
ERRS	4

Notes:

1. A CALL SYSINR(0,0,0,0,0) can be used to prematurely stop the reading from the current file. The next read will then start with the next file in the stack or one specified in a subsequent call to SYSINR.

2. If nc=-99 is specified, all nesting levels are totally cleared.

3. Errors resulting from calls to SYSINR are given only at the time of the next read operation. For example, if an invalid file name is specified, no indication will occur until the program actually attempts to read from that file.

4. Calls to SYSINL and SYSINM after a call to SYSINR without any intervening reads can affect the result obtained from these subroutines.

SYSMSG

This subroutine may be used to control system messages associated with scheduled commands resulting from calls to subroutines SAVREQ and SIGNOF. If the subroutine is called, all system messages associated with the scheduled command (except the final *End message) are suppressed. Messages issued by the program or processor are not suppressed.

Calling Sequence: CALL SYSMSG(1)

TABS

This subroutine is used to specify input or output tab settings from within a program. Its effect is the same as that of a /TABIN or /TABOUT command. The tab settings remain in effect after the end of the job. The physical TAB stops on the workstation are not set by this routine.

Calling Sequence: CALL TABS(num,tab {,&n})

Arguments:

num specifies the number of tab positions being set and must be positive if input tabs are to be set, and must be negative if output tabs are to be set.

tab is an integer array containing the column numbers to be used, in ascending order. These are the same numbers that would be used in a /TABIN or /TABOUT command.

n is a statement number to return to in the event of an error. This argument is optional.

TBAS64

This subroutine converts text to base 64 (3 bytes --> 4 bytes) This is the "base 64" encoding scheme used for e-mail mime, as defined in RFC 1521. It encodes any 8-bit text to printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used. An additional character "=" is used for padding at the end of the output if the input length is not a multiple of 3 (see notes below); this allows the original exact length to be determined when decoding. (This routine is re-entrant.)

See also: FBAS64 - decode from base 64; and UUENC - uuencode-style encode.

Calling Sequence: CALL TBAS64(intxt,inlen,outtxt,outlen)

Arguments:

intxt input data.

inlen length of input data. If 0 or less, outlen is set to 0 and no other action is taken.

outtxt area to receive output data. Size must be at least (n/3)*4 bytes, where n is inlen rounded up to a multiple of 3.

outlen this routine sets outlen to the length of the output data, including pad characters if any. it is always a multiple of 4.

Output pad characters:

If the input length is not a multiple of 3, the last input triple is padded on the right with binary zeros before it is converted, and the last 1 or 2 bytes of the last output quartet are set to the pad character ("="). If input length is 3m+1, there are 2 pad chars; if 3m+2 there is 1 pad char. this lets the decode routine determine the exact length of the original input.

Notes:

1. Other 3-to-4 encoding schemes may use a different translate table and pad character, and may handle lengths which are not a multiple of 3 differently.

2. This routine generates ebcdic (not ascii) printable characters in the output. if ASCII characters are desired, the caller must convert the output to ASCII after calling this routine.

Examples:

```
intxt=x'12', inlen=1:   outtxt="eg==", outlen=4

intxt=x'1234', inlen=2:   outtxt="ejq=", outlen=4

intxt=x'123456', inlen=3:   outtxt="ejrw", outlen=4

intxt=x'12345678', inlen=4:   outtxt="ejrwea==", outlen=8
```

TBIT

This FORTRAN INTEGER function may be used in conjunction with BITON, BITOFF, and BITFLP. It returns the current value of a bit. Your program must declare this function as INTEGER.

Calling Sequence: n=TBIT(a,k)

Arguments:

a is the location of the byte.

k is the position of the bit. *k* may have the values 0,1,2,... The first bit position is referred to as bit 0. For example, if *k* is 24, the value of the first bit of the fourth byte of *a* is returned. If *k* is negative, a value of zero is returned.

TEXTLC

This subroutine turns off the translation of lower case characters to their upper case equivalents for future reads from a workstation during the current job. This command is similar in effect to the /TEXT LC command.

Calling Sequence: CALL TEXTLC

TEXTUC

This subroutine turns on the translation of lower case characters to their upper case equivalents for future reads from a workstation during the current job. This command is similar in effect to the /TEXT UC command. This mode is the default unless changed via a CALL TEXTLC or by a /TEXT LC command.

Calling Sequence: CALL TEXTUC

TIMCON

This subroutine converts the time of day, in the form of a character string or an integer value, from one format to another. The second parameter *arg1* is converted from a format specified in *n1* to a format specified in *n2* and is returned in *arg2*. There are 16 format types listed below. Formats 0 to 13 are character string formats and format 14 and 15 are integer values where time is expressed in seconds and minutes respectively.

When input format type *n1* is 0 (zero), the TIMCON subroutine tries to find a correct time sequence by testing all formats. If a match is found, the correct time is converted into *arg2* in format *n2*.

Calling Sequence: CALL TIMCON(n1,arg1,n2,arg2,irc{,itype})

Arguments:

n1 is the input format type. The format type can be any number from 0 to 15 as defined below.

Type (n1/n2)	Format (arg1/2)	Length	Example
0(arg1 only)	?	12	any below
1	HH	4	15
2	HH?M	8	3 pm
3	HHMM	4	1512
4	HHMM?M	8	312 pm
5	HH.MM	8	15.12
6	HH:MM	8	15:12
7	HH.MM?M	8	3.12 pm
8	HH:MM?M	8	3:12 pm
9	HHMM.SS	8	1512.55
10	HH.MM.SS	8	15.12.55
11	HH:MM:SS	8	15:12:55
12	HH.MM.SS?M	12	3.12.55 pm
13	HH:MM:SS?M	12	3:12:55 pm
14	integer time in seconds	4	54775
15	integer time in minutes	4	912

arg1 is the input character string if type is 0-13, otherwise for type 14 and 15 an integer*4 is expected. The length of the field must be at least as large as is indicated in the table above. When the input type of 0 is specified, the format *arg1* can be any form including "noon" and "midnight" in English, Dutch, French, German, Italian, Portuguese, and Spanish.

n2 is the output format type (see *n1* above). Type 0 is not allowed.

arg2 is the output character string if type is 1-13, otherwise for 14 and 15 an interger*4 is returned. The length of the field must be at least as large as is indicated in the table above.

irc is the return code describing the conversion. The following return codes are possible:

-1	blank input field
1	invalid integer in time specification
2	invalid delimiter
3	time is not inside 0-2400 hours
4	invalid am/pm specified
5	invalid format type *(n1* was not 0-15 or *n2* was not 1-15)
6	invalid format (wrong number of characters for conversion) or when input type is zero, the input field fit no recognizable format.

itype (optional) echoes the input type format when types 1-15 are used. When input type is 0, the format type assumed by TIMCON is reported here. *itype=0* indicates that 1 of the keywords, noon or midnight was used.

Examples

The following examples using input type 0 and output type 11, 13, and 14.

sample input using type 0	output as 13 format	output as 11 format	t in secs 14 format	returned itype
13	01:00:00 pm	13:00:00	46800	1
3 pm	03:00:00 pm	15:00:00	54000	2
512 am	05:12:00 am	05:12:00	18720	4
12:12	12:12:00 pm	12:12:00	43920	6
345.59	03:45:59 am	03:45:59	13559	9
noon	12:00:00 pm	12:00:00	43200	0

TIMDAT

This routine gets the time of day (by TSTIME) and date (by TSDATE). This routine handles the problem of midnight occurring between the calls to TSDATE and TSTIME. It always returns a valid time and date pair.

Calling Sequence: CALL timdat(n,time,m,date1,[date2,]...)

Arguments:

n is the type of time wanted (as in TSTIME).

time is the output time of day.

m is the type of date wanted (as in TSDATE).

date1,date2 are the output dates.

Example:

```
CHARACTER TIME*8,DATE*16
CALL TIMDAT(5,TIME,1,DATE)
```

TIMOFF

A call to this subroutine can be used to display a message giving the computer time used (in service units) since TIMON was called. If however, TIMON was not previously called, the time displayed by TIMOFF is the time from the beginning of execution of the program.

Calling Sequence: CALL TIMOFF

TIMON

A call to this subroutine is used to reset the job time counter used in the TIMOFF subroutine.

Calling Sequence: CALL TIMON

TOLC

This subroutine translates a string of characters to lower case.

Calling Sequence: CALL TOLC(string,strlen)

Arguments:

string is a string of any length to be converted to upper case.

strlen is the length of *string.*

Example:

Given that:

```
STRING --> 'ABC123'
STRLEN --> 6

CALL TOLC(STRING,STRLEN)
```

Returns:

```
STRING --> 'abc123'
STRLEN --> 6
```

TOUC

This subroutine translates a string of characters to upper case.

Calling Sequence: CALL TOUC(string,strlen)

Arguments:

string is a string of any length to be converted to upper case.

strlen is the length of *string.*

Example:

Given that:

```
STRING --> 'abc123'
STRLEN --> 6

CALL TOUC(STRING,STRLEN)
```

Returns:

```
STRING --> 'ABC123'
STRLEN --> 6
```

TPCLSE

A call to this subroutine cancels the effect of a previous call to the TPOPEN subroutine. This subroutine is for ASCII terminals only.

Calling Sequence: CALL TPCLSE

TPOPEN

This subroutine is for ASCII terminals only. When the print line exceeds the terminal line length, MUSIC usually splits the line into two parts. This feature causes problems when users are directing the output to a paper tape. A call to this subroutine suppresses this line splitting.

Calling Sequence: CALL TPOPEN

TRANSL

This routine modifies characters according to a 256-byte translation table. The machine instruction TR (translate) is used.

Calling Sequence: CALL TRANSL(a,len,table)

Arguments:

a represents the characters to be modified.

len is the length in bytes. *Len* may have any value. If it is 0 or less, no action is taken.

table is the 256-byte translation table supplied by the caller. For each of the 256 possible values of a byte in the argument *a* the table must have a replacement byte.

TRIN

This subroutine cancels the effect of a call to the NOTRIN subroutine. Normal terminal input translation is resumed.

Calling Sequence: CALL TRIN

TSDATE

This subroutine returns the current date in one of several forms as specified by the first argument *n.* The date is returned in the remaining argument(s) *x, y...* The calling sequence must contain the correct number of arguments depending on the value of *n:* See also TIMDAT.

Calling Sequence: CALL TSDATE(n,x,y...)

Arguments:

n is a number from 1 to 14 specifying the format of the date.

1 16-byte printable date: WED MAR 06, 1985
2 8-byte U.S.A. format: mm/dd/yy for example 03/06/85
3 8-byte date: *ddmonyyb* for example 06MAR85b (b represents a blank)
4 *x* = integer year (e.g. 1985),
y = integer day of year (e.g. 65)
5 integer day of week, 1 to 7 (Sunday is 1, Monday is 2, etc.)
6 *x* = integer month (e.g. 3),
y = integer day of month (e.g. 6),
z = integer year (e.g. 1985)
7 8-byte European format: dd/mm/yy for example, 06/03/85
8 8-byte sortable: *yymmddbb* for example 850306bb (b represents a blank)
9 8-byte European, no slashes: *ddmmyybb* for example 060385bb (b represents a blank)
10 4-byte packed decimal: 00yydddC for example hex 0055041C (for Mar.6/85)
11 4-byte sortable binary: yymd for example hex 07C10306 (for Mar.6/1985)
12 10-byte sortable: yyyy/mm/dd for example 1985/03/06
13 8-byte sortable: yyyy/ddd for example 1985/065
14 integer in file system date format: integer day+(integer year-1970)*366

x is the variable name where the date is returned.

y is the second variable name where the date is returned and must be specified if *n* is equal to 4 or 6.

z is the third variable name where the date is returned and must be specified if *n* is equal to 6.

TSTIME

This subroutine returns time of day or job execution time as specified by *n.* See also TIMDAT.

Calling Sequence: CALL TSTIME(n,arg)

Arguments:

n is a number from 1 - 8 specifying the format or the time as follows:

1 8-byte time of day in the form HH.MM.SS for example, 15.04.09
2 integer time of day in seconds
3 integer job execution time since the beginning of execution, in units of 1/100 service unit.
4 Integer time of day in units of 1/300 seconds.
5 8-byte time of day in the form HH:MM:SS for example 15:04:09
6 11-byte time of day in the form HH:MM:SS.TH, where TH is hundredths of seconds, for example 15:04:09.73
7 8-byte time-of-day clock, as a 64-bit unsigned integer, obtained by executing a Store Clock (STCK) instruction. Note that the time-of-day clock is the elapsed time since a fixed date in the past.
8 Time-of-day clock, converted to number of microseconds by shifting right 12 bits, as an 8-byte REAL*8 normalized floating-point value. Obtained by a Store Clock (STCK) instruction.

arg is the variable name where the time is returned.

TSUSER

This subroutine returns information about the user as specified by the first argument *n.*

Calling Sequence: CALL TSUSER(n,arg)

or

CALL TSUSER(n,arga,argb) (for n=11 only)

Arguments:

n is a number from 1 to 15 indicating the type of information required.

1 integer workstation type:
0 batch
1 2741,3767 EBCD code
2 2741,3767,CMCST Correspondence code
3 TTY, narrow carriage (72 characters or less)
4 TTY, wide carriage (more than 72 characters)
5 1050
6 3270
99 other
2 First 8 characters of the user's userid. If the userid is longer than 8 characters, the 8th character is returned as plus sign (+) to indicate truncation. See also n=8, 10.
3 4 character terminal number (always "01 ".) *nn*
4 integer TCB number (session number)
5 terminal primary line length (132 for batch)
6 terminal line speed as preset by installation, in bits/sec (0 for batch)
7 1 if terminal accepts 3270 data streams, 0 otherwise. For example, for PCWS, n=7 returns 1 even though n=1 may return 4.
8 16-character userid of the user. It ends in trailing blanks if the userid is less than 16 characters long.
9 Length of the file ownership part of the userid (1 to 16). This is the length of the userid without the subcode, if any.

10 16-character file ownership id of the user. This is the userid without the subcode, if any.
11 3270 screen size: number of rows is returned in arga (e.g. 24); number of columns is returned in argb (e.g. 80).
12 4 bytes of workstation (terminal) information is returned in arg. The first 2 bytes are as set by call to Q3270 subroutine at sign-on, giving 3270 characteristics:
1st byte: all 0 if batch or not 3270-capable.
bit x'40': PCWS connected via an ASCII line.
bit x'20': NET3270.
bit x'10': 3270 using a protocol convertor (e.g. 7171).
bit x'08': 16 colors.
bit x'04': standard 7 colors (extended color).
bit x'02': monochrome, or only the base 4 colors.
2nd byte:
bit x'80': blink (extended attribute) is supported.
bit x'40': reverse video (extended attribute) supported.
bit x'20': underline (extended attribute) is supported.
bit x'08': reply mode: extended field mode supported.
bit x'04': reply mode: character mode supported.
bit x'02': workstation allows entry of DBCS SO/SI chars.
3rd and 4th bytes: reserved.
13 Integer national language number in effect.
=1 English
=2 French
=3 Kanji (Japanese)
=4 Portuguese
=5 Spanish
=6 German
14 Integer national language display mode. Test result by looking at bits.
= 1 if in double byte display mode (ex: Kanji)
15 Integer userid type number.

arg is the variable name where the information is returned.

UUDEC

This subroutine converts text from UUENCODE form (4 bytes --> 3 bytes). This is the UUENCODE encoding scheme used by the UNIX and DOS utility uuencode/uudecode. It encodes any 8-bit text to printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used.

This routine decodes the printable data, back to the original data. (This routine is re-entrant.)

See also: UUENC - encode; TBAS64 - encode to base 64 (mime); and FBAS64 - decode from base 64 (mime).

Calling Sequence: CALL UUDEC(intxt,inlen,outtxt,outlen,retcod,displ)

Arguments:

intxt input data (encoded as by uuencode utility). It is assumed to start on a quartet (group of 4 bytes in the encoded text) boundary. Note: intxt is destroyed by this routine (tranlated in place), so the caller should pass a copy of the original data, if it is to be used again. intxt is destroyed only up to (not including) the first blank or invalid character or to the end of the last full input quartet.

inlen — length of input data. If 0 or less, this routine sets outlen=0, retcod=0, and displ=0 and no other action is taken.

outtxt — area to receive output (decoded) data. It must be large enough to hold the output data. Maximum possible output is ((inlen+1)/4)*3 bytes, where / is integer divide i.e. discard the remainder.

outlen — this routine sets outlen to the length of the output data. This is a multiple of 3. outlen is always the number of bytes stored into outtxt.

retcod,displ — the routine sets these integer arguments to indicate various cases, errors, and ending conditions. retcod is a return code and displ is a displacement within intxt, usually indicating how many input characters have been processed. Scanning stops when a blank is found, or an invalid character is found; this we call the "scan end".

1. if the scan end immediately follows a quartet, retcod=0 and displ=displacement to scan end i.e. 4*(number of quartets processed). outlen=3*(number of quartets processed).
2. if scan end is within a quartet, retcod=number of of chars in last quartet before the scan end i.e. 1 to 3, and displ=displacement to end of the last complete quartet. outlen=3*(number of complete quartets processed). Only the first displ bytes of intxt are destroyed; the partial quartet at the end is intact, and can be concatenated with later data and passed to a subsequent call to this routine.
3. if an invalid character is found, retcod=4 and displ=displacement to the bad character. any preceding complete quartets are processed, and outlen=3*(the number of them).

Notes:

1. intxt argument is modified by this routine. See description of intxt above.

2. this routine assumes the encoded text contains EBCDIC (not ASCII) printable characters. If the input is ASCII, the caller must convert the input to EBCDIC before calling this routine.

UUENC

This subroutine converts text to uuencode form (3 bytes --> 4 bytes). This is the encoding scheme used by the UNIX and DOS utility UUENCODE. It encodes any 8-bit text to (ebcdic) printable characters, producing 4 output chars for each 3 input bytes. see the translate table in this routine for the 64 printable characters used. (This routine is re-entrant.)

See also: UUDEC - decode; TBAS64 - encode to base 64 (mime); FBAS64 - decode from base 64 (mime).

Calling Sequence: CALL UUENC(intxt,inlen,outtxt,outlen)

Arguments:

intxt — input data.

inlen — length of input data. If 0 or less, outlen is set to 0 and no other action is taken. Note: if inlen is not a multiple of 3, the input data is considered to be extended at the end with 1 or 2 binary 0 bytes (x'00'), so as to be a multiple of 3, and the original input length is not recorded in the encoded data.

outtxt — area to receive output data. Size must be at least (n/3)*4 bytes, where n is inlen rounded up to a multiple of 3.

outlen this routine sets outlen to the length of the output data. It is always a multiple of 4.

VERALL

This routine performs a variation of the VERIFY routine. It verifies that all the characters of a string are one of the group of characters specified by a *reference character string*. All non-reference characters located will have their relative position in the string returned, as well as a count of all mismatches.

Calling Sequence: CALL VERALL(string,strlen,refstr,reflen,pos,numpos)

Arguments:

string is a character string of length *strlen* that is to be verified.

strlen is the length of *string.*

refstr is the group of character(s) that constitute the range of acceptable characters that can be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen is the length of the *refstr* in the range of 1 to 256.

pos is an integer array where the relative character positions of all *non-refstr* characters are returned. *pos* should be dimensioned to the byte length of *string,* that is *pos(strlen).* If a character is detected in *string*, that is not one of the reference characters, its relative position is returned in *pos* (i).

numpos is an integer where the number of relative character positions of all non -*refstr* characters is returned.

Example:

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10

CALL VERALL(STRING,STRLEN,REFSTR,REFLEN,POS,NUMPOS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789'
REFLEN --> 10
POS    --> 1, 2, 5, 8, 0, 0, 0, 0
NUMPOS --> 4
```

VERIFY

This routine verifies that all the characters of a string are one of the group of characters specified by a *reference character string*. If a non-reference character is located, its relative position in the string is returned.

Calling Sequence: CALL VERIFY(string,strlen,refstr,reflen,pos)

Arguments:

string — is a character string of length *strlen* that is to be verified.

strlen — is the length of the string, in the range of 1 to 256.

refstr — is the group of character(s) that constitute the range of acceptable characters that can be found in *string.* This character string can be called the *reference string* and the characters *reference characters*.

reflen — is the length of the *refstr,* in the range of 1 to 256.

pos — is an integer returned that is set to 0 if all characters of string are also in the reference string *refstr.* If a character is detected in *string,* that is not one of the reference characters, its relative position is returned in *pos.*

Example:

Given that:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11

CALL VERIFY(STRING,STRLEN,REFSTR,REFLEN,POS)
```

Returns:

```
STRING --> 'bb12t45b'
STRLEN --> 8
REFSTR --> '0123456789b'
REFLEN --> 11
POS    --> 5
```

WORD

This routine parses a string by isolating substrings (words) delimited by a specified character (usually a blank). It returns the number of words or substrings found, as well as their positions and lengths.

Calling Sequence: CALL WORD(string,strlen,number,pos,lens,delim)

Arguments:

string — is a character string to be parsed of length *strlen.*

strlen is the length of the string. When *strlen* =0 no action is taken by WORD.

number is the number of words found in the string.

pos is an array where the relative positions of the words found in *string* are returned. The array dimension should be as large as the expected number of words to be found in *string*. For example, a string of length 80 can only have 40 individual words since the delimiters would occupy 40 bytes.

lens is an array where the relative lengths of the words found in *string* are returned. The array dimension should be as large as the expected number of words to be found in *string*. For example a string of length 80 can only have 40 individual words since the delimiters would occupy 40 bytes.

delim is a one byte string (usually a blank specified as ' ') which serves as the word delimiters. WORD scans *string* and reports, as words, the substrings separated by the delim characters. Consecutive multiple occurrences of the delim character are ignored.

Example:

Given that:

```
STRING --> 'Mary had a little lamb.  '
STRLEN --> 25
DELIM  --> ' '

Note: POS and LEN are dimensioned to 12, for this example.

CALL WORD(STRING,STRLEN,NUMBER,POS,LEN,DELIM)
```

Returns:

```
STRING --> 'Mary had a little lamb.  '
STRLEN --> 25
NUMBER --> 5
DELIM  --> ' '
POS    --> 1 6 10 12 19
LEN    --> 4 3  1  6  5
```

XGCOFF

(Fortran G1 only) A call to this subroutine cancels the effect of a previous call to the XGCON subroutine.

Calling Sequence: CALL XGCOFF

XGCON

(Fortran G1 only) A call to this subroutine permits use of an extended form of G format conversion which permits free format input. The extended form remains in effect until a subsequent call to XGCOFF is made.

In the extended form, the field width specified is used as a maximum field width. However, the field width is considered smaller if the number ends before the specified field width. If no number is found within the

specified field width, then it is set to 0.

Calling Sequence: CALL XGCON

Example:

This facility allows an input line to be scanned totally in free format or parts can be in fixed format as well, as in the following example.

```
    CALL XGCON
  1 READ(9,2)A,I
  2 FORMAT(2G80.0)
```

The format statement is interpreted as meaning that the two numbers expected may appear anywhere on the data statement, as long as they are separated by at least one blank. Any number not found on the data statement is assumed to be zero.

X2C

This routine converts hexadecimal strings to character equivalents (EBCDIC).

Calling Sequence: CALL X2C(hexstr,hexlen,string,irc,badchr)

Arguments:

hexstr — is a character string of length *hexlen* which has in it only hexadecimal digits 0-9 and A-F. These must be paired, e.g. F1F2F3.

hexlen — is the length of the *hexstr*. This length must be an even integer.

string — is the character string to be returned, in the range of 1 to 256. The string should be at least half the length of *hexstr*.

irc — is the return code describing the conversion.
- irc=-1 the HEXSTR length was 0.
- irc=0 conversion was successful.
- irc=1 an illegal character was found in the string.
- irc=3 conversion not possible, odd number of hex digits were found.

Note: When *irc* is not 0 *string* remains unchanged by X2C.

badchr — a one byte variable where the bad character (characters not in the range A-F, 0-9) is returned when *irc=1*.

Example:

Given that:

```
HEXSTR --> 'C1C6F3F4F5F6F7F8F960'
HEXLEN --> 10

CALL X2C(HEXSTR,HEXLEN,STRING,IRC,BADCHR)
```

Returns:

```
HEXSTR --> 'C1C6F3F4F5F6F7F8F960'
HEXLEN --> 10
STRING --> 'AF123456789/'
IRC    --> 0
BADCHR --> 'b'
```

X2I

This routine converts a hexadecimal character string to a four byte integer value.

Calling Sequence: CALL X2I(hexstr,hexlen,i,irc,badchr)

Arguments:

hexstr — is a character string of length *hexlen* which has in it only hexadecimal digits 0-9 and A-F.

hexlen — is the length of the *hexstr*. This length must be an integer less than or equal to 8 bytes.

i — is the integer to be returned.

irc — is the return code describing the conversion.
irc=-1 the HEXSTR length was 0.
irc=0 conversion was successful.
irc=1 an illegal character was found in the string.
irc=2 conversion not possible, length of *hexstr* was greater than 8.

Note: If *irc* is not 0, then: i is set to zero.

badchr — a one byte variable where the bad character (characters not in the range A-F, 0-9) is returned when *irc=1*.

Example:

Given that:

```
HEXSTR --> 'E0E'
HEXLEN --> 3
```

```
CALL X2I(HEXSTR,HEXLEN,I,IRC,BADCHR)
```

Returns:

```
HEXSTR --> 'E0E'
HEXLEN --> 3
I      --> 3598
IRC    --> 0
BADCHR --> 'b'
```

ZERO

This subroutine sets bytes of main storage to zero. The user should use this routine to zero large sections of storage (eg. arrays with more than 500 elements).

Calling Sequence: CALL ZERO(a,m {,n})

Arguments:

a is the starting location of the bytes in main storage.

m is the number of words to turn to zero. If *m* is less than 1 no action is taken.

n is the length of each word. If *n* is omitted, 4 is assumed so that the subroutine sets *m* full words of main storage to zero.

Dynamic Access to Files

FORTRAN programs on the MUSIC system can read and write files without having to predefine the file names on /FILE statements. This dynamic style of access allows more program flexibility. For example, at execution time a program can prompt the user to specify the name of a file to be read or written. Input/output is done using the standard FORTRAN statements (READ, WRITE, REWIND, ENDFILE, etc.)

Dynamic access is controlled by four system subroutines: OPNFIL (open a file), CLSFIL (close a file), SETINF (specify information about a new file to be created), and FILMSG (get error message text corresponding to an error number). Some usage examples are given at the end of this article.

OPNFIL Subroutine (Open a File)

This routine associates a file name with a FORTRAN unit number (1 to 15 for FORTRAN G1, or 1 to 99 for VS FORTRAN. Both the file name and the unit number are specified by the user. Various option keywords may also be specified, indicating a new or existing file, read or write access, etc. The system attempts to open the file, and passes a return code back to the user. A zero return code indicates a successful open, and the user may then do FORTRAN I/O operations on the unit number, just as if the unit had been defined by a /FILE statement.

An 8-character MVS DDname (data definition name) can be specified instead of a unit number. In that case, the DDname is added to the job's DDname table (if not already there) and it is associated with the SL file. This allows programs running in OS simulation mode (such as PL/I, COBOL and VS ASSEMBLER programs) to access files dynamically. OPNFIL gives error code 31 if it is unable to define the DDname because the DDname table is full.

When a program running in OS simulation mode calls OPNFIL with a unit number, the corresponding FORTRAN DDname (FTnnF001) is also defined, in addition to the unit number. This feature allows VS FORTRAN programs to use OPNFIL. Refer to the section on VS FORTRAN for further information.

CLSFIL Subroutine (Close a File)

The CLSFIL routine closes a unit number previously opened by OPNFIL. The system closes the corresponding file, and the unit number becomes undefined. The unit may be redefined by a later call to OPNFIL. CLSFIL gives a return code to the user. Zero indicates a successful close.

A 8-character MVS DDname can be specified instead of a unit number. This would be a DDname previously defined by a call to OPNFIL.

SETINF Subroutine (Set Information for a New File)

When a new file is created by OPNFIL, the attributes of the file must be supplied. These include file size, record length, record format, and access control (public, private, etc.) They are specified by calling SETINF before calling OPNFIL. SETINF stores the information in a common block which is then available to OPNFIL.

FILMSG Subroutine (Get Error Description)

When an open or close is unsuccessful, a nonzero return code indicates the reason for the error. The error code is typically a 2-digit number. If the program wishes to display an error message at this point, descriptive text corresponding to the error code can be obtained by calling FILMSG. Optionally, FILMSG can display a message itself (including the unit number) and terminate the job.

Calling Sequences

Some of the arguments may be omitted, as indicated in the descriptions below. Note that all numerical arguments are of type INTEGER*4.

Calling Sequence: CALL OPNFIL(unit,retcod,'shortname ','options.')

or

CALL OPNFIL(unit,retcod,-1,'longname ','options.')

or

CALL OPNFIL(UNIT,retcod,TYPE)

Arguments:

unit — FORTRAN unit number (1 to 99), or an 8-character DDname enclosed in single quotes. For FORTRAN G1, the unit number must be 1 to 15.

retcod — Error number returned by the routine. Zero means successful open.

'shortname ' — The name of the file to be opened. It may include the userid prefix (usrd:). A blank must follow the name, unless it is the maximum length (22 characters). Special names '&&TEMP ' (temporary new file) and '/INPUT ' (MUSIC Input File) may be used. If a temporary file is to be kept, use the RENAME or REPL option when closing it.

'longname' — Same as shortname except that the maximum length of the file name is 64 characters (can include directory prefixes). The option "-1" must be present in the calling sequence before the longname.

'options.' — (This argument is optional.) This is a character string containing option keywords. The keywords are separated by one or more blanks and the last one is followed by a period. They are described below. If this argument is omitted, 'OKOLD RDOK.' is assumed, i.e. an existing file is to be read.

TYPE — A special unit identifier may be specified instead of a file name. The allowable values are:

TYPE = 0 Undefined file.
5 Input stream (card reader if batch).
6 Printed or displayed output.
7 Punched output.
8 Dummy file (read gives EOF, writes ignored).
9 Conversational reads.
10 Special holding file (output).

Calling Sequence: CALL CLSFIL(unit,retcod,'options.')

or

CALL CLSFIL(unit,retcod,'options.','newname ')

Arguments:

unit — FORTRAN unit number (1 to 99), or an 8-character DDname enclosed in single quotes. For FORTRAN G1, the unit number must be 1 to 15.

retcod — Error number returned by the routine. Zero means successful close.

'options.' — (This argument is optional.) Keyword options, in the same format as for OPNFIL. The keywords for close are described below.

'newname ' — This optional last argument is used only when the RENAME or REPL option is specified. It is the new name to be given to the file.

Calling Sequence: CALL SETINF(PRIMSP,SECSP,LRECL,RECFM,'options.')

Arguments:

PRIMSP — Initial space to be allocated for the new file, in units of 1K = 1024 bytes. If the argument is omitted or 0, 32 is assumed.

SECSP — Secondary space to be allocated whenever more space is needed in the file. It is specified either as an absolute amount (in units of 1K = 1024 bytes) or as -n, meaning n% of the existing space. If the argument is omitted or 0, 50% is assumed.

LRECL — Logical record length of the file, 0 to 32760. If omitted, 80 is assumed. Specify 0 if the record format is V or VC.

RECFM — Record format, as a 2-character item:
'FC' Fixed-length compressed (the default).
'F ' Fixed-length uncompressed.
'VC' Variable-length compressed.
'V ' Variable-length uncompressed.
'U ' Undefined.
If the argument is omitted or 0, 'FC' is assumed.

'options.' — Keywords defining the access control options for the file, in the same format as the options for OPNFIL. The keywords are described below. If omitted, 'PRIV.' (private file) is assumed, i.e. only the owner may read or write the file.

Calling Sequence: CALL FILMSG(errcod,buffer,length)

or

CALL FILMSG(errcod,unit)

Arguments:

errcod — An error number returned by OPNFIL or CLSFIL.

buffer — An array to receive the error description.

length — The length of BUFFER, in bytes. A length of 70 or more is adequate, but any length can be

specified. Unused bytes are set to blanks by the routine.

unit — If FILMSG is called with only two arguments, the second is assumed to be the unit number associated with the error. If *errcod* is nonzero, a message is displayed (containing the error number, unit number, and error description) and the job is terminated. If *errcod* is 0, no message is displayed and the job continues.

Option Keywords for Open

OKOLD — The file may already exist.

OKNEW — A new file may be created. SETINF must be called to define the attributes of the new file. Note: at least one of OKOLD, OKNEW must be used.

RDOK — Read operations will be allowed.

WROK — Write operations will be allowed.

APPOK — Append (adding to the end of the file) will be allowed. The file will be positioned to the end of existing data. WROK must also be used.

POSEND — Position to the end of the file's data.

ENQSHR — Force enqueue for shared control of the file. Normally shared control is used only when WROK is not specified.

ENQEXCL — Force enqueue for exclusive control of the file.

Option Keywords for Close

RLSE — Release any unused space. This option is recommended if writes were done to the file, especially when creating a new file.

RENAME — Rename the file to the name specified in the 'newname' argument.

REPL — Rename the file to the name specified in the 'newname' argument. This is similar to RENAME, except that if a file already exists with the new name, it is deleted.

DEL — Delete the entire file.

TCLOSE — Specifies that the CLOSE function is to be performed but that the file is to remain open for further requests. The current end of data pointer is updated in the file's directory. RLSE is the only other option that can be given when TCLOSE is used.

CTAG — Set a new tag field when the file is closed.

Note: For RENAME and REPL, new access control options are used for the new file, and these should be specified by calling SETINF before calling CLSFIL. In this case, the first 4 arguments of SETINF may be specified as 0.

Keywords for SETINF Access Control Options

Access is defined for two classes of users: (1) an owner of the file (the user's userid matches the file's userid), and (2) non-owners (all other users). The default is a *private* file, meaning that an owner has unrestricted access while non-owners have no access at all. The suffix "(OWN)" indicates that the keyword applies to access by an owner.

PUBL	A publicly readable file. Non-owners may read the file but not modify it. Also, the file is placed in the common index (see COM below).
PRIV	A private file (the default). Non-owners can neither read nor write the file.
SHR	Non-owners may read the file.
COM	The file is placed in the common index, so that non-owners can refer to it without having to include the file's userid in the file name.
RD	Same as SHR.
NORD	Non-owners may not read the file.
WR	Non-owners may modify the file.
NOWR	Non-owners may not modify the file.
XO	The file is execute-only for non-owners. Execute-only means that the file may be executed as a program, but not read as data.
AO	The only type of write access for non-owners is append. This means that non-owners may add data to the end of the file, but not overwrite existing data. This is useful for *log* files.
RD(OWN)	Owners may read the file.
NORD(OWN)	Owners may not read the file.
WR(OWN)	Owners may modify the file.
NOWR(OWN)	Owners may not modify the file.
XO(OWN)	The file is execute-only for owners.
AO(OWN)	The only type of write access for owners is append.

The following are assumed by default: NORD, NOWR, RD(OWN), WR(OWN). The specification 'PUBL.' is equivalent to 'RD COM.'

Common Blocks Used

Several named common blocks are used for communication among the calling program and the system subroutines. For most applications, the user need not be aware of them.

/FILINF/	(20 bytes) This area contains information about a new file to be created (set by SETINF

subroutine), and also receives information when an existing file is opened.

/FILTAG/ (64 bytes) This is the 64-character tag field. To assign a tag to a new file, place the tag in this area before calling OPNFIL, but after calling SETINF. Any call to SETINF zeros this area. Also, when an existing file is opened, its tag is put here.

/FILEFP/ (10 bytes) End-of-file information is placed here when a file is opened.

/FILRGX/ (12 bytes) When OPNFIL or CLSFIL is called, the first 12 bytes of the system request argument (after the request is issued) are placed here. The 10th byte has bit X'80' on whenever a new file is created. After a close with RENAME or REPL, this bit is off if an existing file was replaced.

Error Codes and Descriptions

```
 1    END OF DATA SET ENCOUNTERED
 2    INCORRECT LENGTH
10    INVALID REQ
11    INVALID REQ PARAMETER
12    FILE NAME INVALID
19    INVALID ARGUMENTS IN CALL TO SERVICE SUBROUTINE
20    TOO MANY OPEN FILES
21    NOT YOUR LIBRARY
22    NOT YOUR FILE
23    VIOLATION OF WRITE RULE
24    ATTEMPT TO READ BEYOND END OF WRITTEN INFO
25    WRITE THEN READ SEQ INVALID
26    YOUR USERID CANNOT CREATE FILES ACCESSIBLE BY OTHERS
27    YOUR USERID CANNOT CREATE FILES IN THE COMMON INDEX
30    FILE NOT FOUND
31    DDNAME NOT FOUND
      (For a call to OPNFIL to define a DDname, error 31
      means there is no more room in the DDname table.)
32    FILE ALREADY EXISTS
33    FILE IN USE
34    COMMON NAME USED BY SOMEONE ELSE
35    UNIT NUMBER NOT DEFINED
36    SUBDIRECTORY DOES NOT EXIST
40    SPACE QUOTA EXCEEDED FOR THIS USERID
41    SPACE QUOTA EXCEEDED FOR THIS FILE
42    CANNOT ADD SPACE TO THIS FILE
43    REQUESTED ACCESS OR OPERATION NOT ALLOWED
44    REQ BEYOND EXTENT OF FILE
45    FILE RECFM NOT DEFINED
46    FILE CANNOT BE READ SEQUENTIALLY
47    INSUFFICIENT SPACE FOR BUFFER ALLOC
48    MIN RECORD LEN IS 80 FOR THIS FILE TYPE
50    FILE NOT ONLINE
51    NOT ENOUGH FREE DISK SPACE
52    NOT ENOUGH FREE DISK SPACE (INDEX)
60    RD I/O ERROR IN FILE
61    WR I/O ERROR IN FILE
62    RD I/O ERROR IN SYSTEM AREA
63    WR I/O ERROR IN SYSTEM AREA
```

```
64    INDEX IN ERROR
65    HEADER IN ERROR
66    MAP INTEGRITY ERROR
67    INDEX/HEADER MISMATCH
70    SYSTEM FILE ERROR
```

Examples

1. This example opens an existing file JULY.DATA as unit 1, reads it, and closes it.

```
CALL OPNFIL(1,K,'JULY.DATA ')
CALL FILMSG(K,1)
READ(1,...   (read the file)
...
CALL CLSFIL(1,K)
```

2. This sample program opens a new temporary file as unit 12, with a size of 100K, and writes some data to it. It then closes the file, renaming it to a file name supplied by the user. Any unused space is released. The new file is made public.

```
      LOGICAL*1 NAME(22)
      INTEGER UNIT/12/
      CALL SETINF(100)
      CALL OPNFIL(UNIT,K,'&&TEMP ','OKNEW WROK.')
      CALL FILMSG(K,UNIT)
      WRITE(UNIT,10)
10    FORMAT('DATA1...'/'DATA2...')
      WRITE(6,20)
20    FORMAT(' ENTER NAME OF NEW FILE')
      READ(9,30) NAME
30    FORMAT(22A1)
      CALL SETINF(0,0,0,0,'PUBL.')
      CALL CLSFIL(UNIT,K,'RLSE RENAME.',NAME)
      CALL FILMSG(K,UNIT)
      WRITE(6,40)
40    FORMAT(' SAVED')
      STOP
      END
```

QFOPEN, QFCLOS, QFREAD, QFBKRD, QFRBA, QFREW

These routines allow quick record-by-record read of a sequential file. Disk reads are done by UIO request, several blocks at a time. QFREAD deblocks and decompresses the logical records itself, thus avoiding the overhead of SVC and MFIO processing for each logical record. The file's record format must not be U.

QFBKRD reads the previous logical record. Currently, only supported for RECFM=FC,VC files.

Note: These routines are re-entrant (read-only)

QFOPEN - Open a file and define buffer area

QFOPEN opens the file (by name or ddname, etc.). The caller provides a work area, where the routines store control information and UIO buffers. The same work area must be used in all subsequent calls for that file, until the file is closed by QFCLOS. The caller should not change the contents of the work area between calls. Each concurrently open file needs its own work area.

Calling Sequence: CALL QFOPEN(retcod,wkarea,wklen,shortname,options,
phys,info,eofp,tag,uinf,xinf)

Calling Sequence: CALL QFOPEN(retcod,wkarea,wklen,-1,longname,options,
phys,info,eofp,tag,uinf,xinf)

The "options" argument is optional (can be omitted) unless phys, etc. are present. phys,info,... are optional and are the areas to be used in place of the common block areas /QFPHYS/, etc. All these arguments must be present if the routine is part of a re-entrant module, to avoid storing into the common blocks (which are part of the module). The arguments are described in detail below.

QFCLOS - Close a file and realease buffer area

Calling Sequence: CALL QFCLOS(retcod,wkarea)

QFREAD - Read next logical record

Calling Sequence: CALL QFREAD(retcod,wkarea,recbuf,lenbuf)

QFBKRD - Read previous logical record

Calling Sequence: CALL QFBKRD(retcod,wkarea,recbuf,lenbuf)

Currently only supports RECFM=FC,VC files. Give EOF (RC=1) if attempt to read backwards too far. You can use the QFRBA to set a very high number for the RBA and the next call to QFBKRD will read the last record in the file.

QFRBA - Set RBA (Relative Byte Address) for next read

Calling Sequence: CALL QFRBA(retcod,wkarea,rba)

QFREW - Rewind the file

Calling Sequence: CALL QFREW(retcod,wkarea)

Equivalent to call QFRBA(retcod,wkarea,512 or 514).

Arguments

retcod — MFIO return code. These routines may set return code 19 ("invalid arguments in call to service subroutine") for invalid calls, e.g. work area too small or not initialized by open.

wkarea — The work area for the file. See QFWRK DSECT. Contains control information followed by 1 or more 512-byte buffers. Must be on a fullword or doubleword boundary. Minimum size is about 700 bytes. Recommended size (for maximum speed) is about 10k+200. Maximum size is (2**24)-1. A very big work area should be avoided, unless the entire file will fit into the buffer area, because each I/O tries to read the greatest number of buffers possible and this could cause excessive I/O for random access in a big file.

wklen — Length of the work area, in bytes.

shortname
or
-1,longname — File name (or ddname, etc.) for the open. Maximum length is 22 for a short name and 64 for a long name. If an actual file name is provided and it is shorter than the maximum, it must be followed by a blank.

If the "LU" option is used (see below), the file name is actually a 4-byte logical unit number.

Special provision for using a file that is already open: If the options argument contains the keyword "IU", then the name argument is a 4-byte internal unit number (IU), and the file is assumed to be already open. Other option keywords are ignored if present. At the time of the call to QFOPEN, common blocks /QFINFO/ and /QFEIOP/ (or their replacements) must contain data as set by the original open. For the IU option, QFOPEN does not do an open, so it does not fill in any common blocks.

options — (Optional) a character string of the form 'keyword keyword ... keyword.' The keywords are separated by blanks and the last one is followed by a period. String '.' specifies no option keywords. The keywords specify MFIO options for the open:

DDNAME
DDORDS
MEMBER
LU
NODATE
ENQEXCL
PGMP
IU (see above)

OPEN, OKOLD, RDOK
(these are allowed but ignored; this gives compatibility with MFACT) In all cases, the open uses MFIO options okold and rdok.

phys,info,eofp,tag,uinf,xinf
These optional arguments specify replacement areas for the common blocks /QFPHYS/,/QFINFO/,etc. The phys argument must be 8 bytes long, the info argument 20

bytes long, etc. When an argument is present, it is used instead of the corresponding common block area. This is useful for cobol programs (which cannot access common blocks) and for a re-entrant module (where the common blocks are part of the module and must not be used).

recbuf — Buffer to receive the logical record. Truncation or blank padding occurs if needed.

lenbuf — The length of the caller logical record buffer (recbuf). it may be any length, including zero.

rba — The MFIO RBA (Relative Byte Address) of the next record to be read. use of qfrba is optional.

Common Blocks Defined:

/QFINFO/	(20 bytes) infout from the open.
/QFTAG/	(64 bytes) tag from the open.
/QFUINF/	(46 bytes) uinfo from the open.
/QFXINF/	(40 bytes) xinfo from the open.
/QFEOFP/	(10 bytes) eofpt from the open.
/QFPHYS/	(8 bytes) phys set after each call to QFREAD.

Note: If QFOPEN provides a replacement area for a common block, that area is used instead, and the common block is not modified.

ITS Subroutines

MUSIC's Indexed Text Search (ITS) facility provides an efficient method of locating words in large documents. The ITSRET utility provides a way to locate and display the results of a search. The subroutines described in this section provide an alternate to ITSRET. The subroutines allow you to write programs that call the search routines and check the results directly.

These routines require a work area. This work area can be obtained dynamically by a call to ITSFID or a work area can be supplied by a call to ITSFIW.

The search routines require access to an index data set built by the ITSBLD or ITSBLD2 utility. The calling sequences allow for multiple index and text data sets, however, currently only one "search set" is supported.

ITSFID

This subroutine initializes using dynamic (Getmained) storage.

Calling Sequence: CALL ITSFID(rc,mhits,nss)

Arguments:

rc 4 bytes are returned:
=0 if ok
=1 if not enough storage available
=2 number of search sets is invalid

mhits (4 bytes) is the maximum number of hits to handle. This routine does a Getmain for the work area. Intermediate results also use this area so make it quite large. Typically 2,000 or 4,000. Each item needs 24 bytes. So MHITS=2,000 needs 48,000 bytes.

nss is the maximum number of search sets (4 bytes). Should be = 1.

ITSFIW

This subroutine initializes using caller's work area.

Approximate amount needed is about 6,000 for fixed work space that supports 1 search set. Add to this 24 bytes per hit wanted. The fixed work space may increase in future to 10,000 or more.

Calling Sequence: CALL ITSFIW(rc,lhits,nss,work,lwork,mhits)

Arguments:

rc 4 bytes are returned:
=0 if ok
=1 if not enough storage available
=2 number of search sets is invalid
=3 Workarea not on double word boundary.

lhits is the limit to number of hits to handle (4 bytes). May not be able to do this amount due to size of input work area. If number set to -1, then take as many as will fit in area.

nss is the maximum number of search sets (4 bytes). Should be = 1.

work is the input work area location (variable length). Must be on double word boundary.

lwork is the length of work area in bytes (4 bytes).

mhits 4 bytes are returned for the number of hits that will be done. It will be the maximum number that fits in area or number as limited by *lhits.*

ITSFOP

This subroutine opens a search set.

Calling Sequence: CALL ITSFOP(rc,ss,fnidx,fntext,debidx,debtext,endmark,endml)

Arguments:

rc 4 bytes are returned:
=0 If ok
=1 If Search Set number invalid or in use.
=2 If Index data set not created by right version of ITSBLD.
=3 Problem opening index component.
MFIO error code 256 added to return code.
=4 Problem opening text component.
MFIO error code 256 added to return code.
=5 Problem reading index component.
MFIO error code 256 added to return code.

ss is the search set number (4 bytes). Must be = 1.

fnidx is the 64 character file name of the index, or is the 1 word *deb* number (format used if already opened).

fntext is the 64 character file name of the text, or is the 1 word *deb* number. Format used if:
a) file already open by caller
b) =0 if text file will not be used by these subroutines.

debidx is the 1 word *deb* number of the index file returned after open.

debtext is the 1 word *deb* number of the text file returned after open.

endmark is the 8 character item ending string used on this search set (returned).

endml is the 4 byte count of the ending marker (returned).

ITSFSS

This subroutine searches for a character string.

Calling Sequence: CALL ITSFSS(rc,sstr,lsstr,nhits)

Arguments:

rc 4 bytes are returned:
=0 OK, something found.
=1 No hits found. There is further information in 3rd byte of RC.

1 Only stop words looked for
2 Hits found but after boolean logic nothing left
3 Not even intermediate hits found

=2 Too many items found. Some hits missing.

=3 Problems with search string. There is further information in 3rd byte of RC.
1 Search string all blanks
2 Search word greater than maximum length (of 64)
3 Search string has quotes, periods
or other special chars around them.
4 Syntax errors

=4 Input search string length too long.

=5 Problem reading index component. MFIO error code 256 added to return code.

sstr is the search string.

lsstr is the length of search string (maximum is 256).

nhits is the number of hits found (returned).

ITSFOR

This subroutine re-orders the matches of the last search. resulting list.

Calling Sequence: CALL ITSFOR(rc,so)

Arguments:

rc 4 bytes are returned:
=0 if ok
=1 invalid sort order request

so is the sort order combination of the file order and weight order wanted. Add the following options to give the sort order.
0 order in file (default if ORDER was not called after last search).
1 reverse order of file (last found item will be first).
2 weighted order (highest weight first).
4 reverse weighted order (highest weight last).

ITSFRE

This subroutine retrieves results.

Calling Sequence: CALL ITSFRE(rc,hn,buff,nhits,layout,txtl,nent)

Arguments:

rc 4 bytes are returned:
=0 OK
=1 Starting hit number too large.
=1 I/O error reading from text component.
MFIO error code 256 added to return code.
(Can also include end-of-file.)
=2 Problem opening text component.
(Delayed open option used.)

MFIO error code 256 added to return code.

hn is the hit number to start with. First one is number 1.

buff is the buffer to return results.

nhits is the number of hits wanted.

layout is the layout of each return item, made up of the following optional fields. Each field will be length 4 if selected and will be in the following order:

Search set number
RBA
Weight
One line of text from text file
Length of text line

Calculate a number made up of the sum of the following wanted.

1 Return search set number
2 Return RBA
4 Return weight
8 Return length of text string. This length is obtained from the file system and may be greater than the number in *txtl*. This option should not be selected if *txtl=0*.

txtl is the maximum length of text from the text file to be read. If 0, nothing will be read. If non-zero, then use this length to read 1 logical record from the text file starting at the RBA. If the record is less than this length pad with blanks only if return length field is not given.

nent is the number of entries returned.

ITSFCL

This subroutine closes the search.

If close all request, then storage is freed and a call to ITSFID or ITSFIW will be required if more searching wanted.

Calling Sequence: CALL ITSFCL(rc,ss)

Arguments:

rc 4 bytes are returned:
=0 if ok

ss is the search set number to close or =0 for all. If search set number given as negative number then just remove search set from list but do not close it.

Sample Program - ITS Subroutines

```
/LOAD VSFORT
C  SAMPLE PROGRAM SHOWING THE USE OF SOME OF THE ITS SUBROUTINES

C  (REAL PROGRAMS SHOULD CHECK RETURN CODES AFTER ALL THE SUBROUTINE
```

```
C   CALLS)

      IMPLICIT INTEGER (A-Z)
C  VARIABLES TO HOLD FILE NAMES
      CHARACTER *64 FNIDX,FNDAT
C  VARIABLE TO HOLD ENDING STRING
      CHARACTER *8 ENDM
C  VARIABLE TO HOLD SEARCH STRING
      CHARACTER *80 SSTR
C  VARIABLE TO HOLD TEXT LINE FROM FILE
      CHARACTER * 80 BUFFER

C -- INITIALIZE USING DYNAMIC STORAGE AREA AND ALLOW FOR 2000 HITS.
      CALL ITSFID(RC,2000,1)
      WRITE(6,100)MHITS
100   FORMAT(' INITIALIATION CALL.  MHITS=',I6)

C -- OPEN THE SEARCH SET
C  SPECIFY FILE NAMES FOR WORD INDEX AND DATA COMPONENT
      FNIDX='$MAN:UR.WIDX'
      FNDAT='$MAN:UR.TOTAL'
      FLGS=0
      CALL ITSFOP(RC,1,FNIDX,FNDAT,DEBIDX,DEBDAT,FLGS,ENDM,ENDML)
      WRITE(6,110)DEBIDX, DEBDAT,ENDM,ENDML
110   FORMAT(' OPEN SS.  DEBNUM=',2I4,' ENDMARK=',A,' L=',I2)

C -- DO A SEARCH.  (SEARCH STRING GIVEN IN VARIABLE SSTR)
      SSTR='SENDFILE'
      LSSTR=80
      CALL ITSFSS(RC,SSTR,LSSTR,NHITS)
      WRITE(6,120)NHITS
120   FORMAT(' SEARCH FOUND', I5, ' HITS')

C -- RETURN THE FIRST LINE OF EACH SECTION THAT HAD A MATCH.
C    (TO READ MORE OF SECTION, ASK FOR RBA AND USE QFREAD OR MFIO
C     SUBROUTINES.)

      DO 40 HN=1,NHITS
      LAYOUT=0
      CALL ITSFRE(RC,HN,BUFFER,1,LAYOUT,80,NENT)
      WRITE(6,140)BUFFER
140   FORMAT(1X,1A80)
40    CONTINUE

      CALL EXIT
      END
```