COMPUTERVISION

CGOS

CGOS 200 GNA
Programmer Reference

Document control number: 19- 00845

Name _____

CGOS 200 GNA
Programmer Reference

Order No. 001-00019
Price $50.00

TABLE OF CONTENTS

Page

Section One
INTRODUCTION 1-1

NOTATION 1-2
ON-LINE DOCUMENTATION 1-2
Access 1-2
Levels 1-3
Listing Documents 1-3
Special Commands 1-4

SectionTwo
CREATING EXECUTABLE CODE 2-1

FORTRAN/S 2-2
Special Consideration 2-2
Arrays 2-2
COMPILATION 2-2
Compiling Routines 2-2
Symbol Map 2-5
LOADING CODE 2-7
Creating The Make File 2-7
Loader Keywords 2-8
Running The Loader 2-15
Loader Glossary 2-18
Loader Errors 2-18
LOADLIB 2-24
LISTLOAD 2-25
GENINDX 2-28
CROSSREF 2-29
SYSTEM REFERENCES 2-30
Symbolic References 2-30
Explicit References 2-30
Symfiles 2-31
EXECUTING APPLICATION CODE 2-31
Coremap 2-32
Sample Bundled Makefile 2-33
Sample Unbundled Make File 2-34
Executing Code At O/S Level 2-36

TABLE OF CONTENTS (Continued)

Page

COMMAND EXECUTION 2-36
Command Table Search 2-36
Sequence of Execution 2-37
Cross-Referenced Command Processing 2-37

Section Three
DEBUGGING 3-1

SUBROUTINE TRACE 3-1
Tracing and Debugging 3-1
Using the Trace Tool 3-1
Memory Requirements 3-1
Controlling Trace Output 3-1
SYMBOLIC REFERENCES 3-2
Symbol Filenames 3-2
Overlays 3-3
Sample Output without Symbolic References 3-3
Sample Output with Symbolic References 3-3
APPLICATION DEBUGGER 3-4
To Access the Debugger 3-4
Features 3-4
Restrictions 3-5
DEBUGGER COMMANDS 3-6
General Control 3-6
Toggles 3-6
Breakpoints 3-7
Controlling the Execution of Breakpoints 3-7
Dump Commands 3-7
Setting Memory 3-7
Searches 3-8
Miscellaneous 3-8
Local Variables 3-8

Section Four
BRIEF SUBROUTINE DESCRIPTIONS 4-1

LINKAGE FOR ROUTINES 4-1
System Resident Routines 4-1
System Overlay Routines 4-1
System Library Routines 4-1

TABLE OF CONTENTS (Continued)

Page

ROUTINES LISTED BY FUNCTION 4-2
File Manipulation 4-2
File Utility Packages 4-2
GETFILE 4-2
PUTFILE 4-3
READFILE 4-3
WRITFILE 4-3
READBFIL 4-3
WRITBFIL 4-4
Processing A Catalog 4-4
Deleting A File 4-4
Filename Manipulation 4-4
Memory Page Manipulation 4-4
Date And Time 4-5
Task-Related Functions 4-5
Errors 4-6
Peripheral Devices 4-6
Subroutine Trace 4-7
Scanning 4-7
Input/Output 4-8
Breaks And Labels 4-9
Double Precision Integers 4-10
Comparison 4-10
Conversion 4-11
Array Manipulation 4-11
Stack Manipulation 4-12
Bit Manipulation 4-12
Sorting Records 4-13
Miscellaneous 4-13

Section Five
SUBROUTINES 5-1

ROUTINE DESCRIPTION CONVENTIONS 5-1
Syntax 5-1
Input 5-2
Output 5-2
Function Return 5-2

TABLE OF CONTENTS (Continued)

Page

UNIVERSAL 5-2
Bit Numbering 5-2
Array Indexing 5-2
PARTICULAR 5-3
FM Name Format 5-3
Options 5-3
File Status Block 5-3
FILE MANAGEMENT 5-4
SUBROUTINE DESCRIPTIONS 5-4
ADJUST 5-5
ALLOPG 5-5
ATTACH 5-6
CARDIN 5-7
CATBREAK 5-8
CATWALK 5-9
CHKBREAK 5-13
CHKINTR 5-14
CHKPROT 5-15
CHKQUIT 5-16
CHKSTOP 5-17
CHKSUM 5-17
CHKTRACE 5-18
CHKUPROT 5-18
CKPPTD 5-19
CKRPTD 5-19
CLEARCOM 5-19
CLOSE 5-20
CLRBIT 5-21
CLRBREAK 5-22
CMBYTF 5-22
CMBYTT 5-23
CMPDAT 5-23
COMPN 5-24
COMPNAM 5-25
COMPUS 5-25
COPEN 5-26
COPYFILE 5-31
DBADD 5-32
DBCMPR 5-32

TABLE OF CONTENTS (Continued)

Page

DBDCR 5-33
DBDIV 5-33
DBHEXLST 5-34
DBHEXBCD 5-35
DBHXNM 5-35
DBINC 5-36
DBINT 5-36
DBINTBCD 5-37
DBINTLST 5-37
DBLE 5-38
DBLSH 5-38
DBMAX 5-39
DBMIN 5-39
DBMUL 5-40
DBNEG 5-40
DBRSH 5-41
DBSUB 5-41
DELETE 5-42
DELETEXT 5-43
DELFIL 5-44
DFLOAT 5-44
DFLOATL 5-45
DIVUS 5-45
ELAPTIME 5-46
ERROR 5-47
FILL 5-48
FILLBYTT 5-48
FILLCHRT 5-49
FILLDB 5-49
FILLF 5-50
FLBCD 5-50
FLOATL 5-51
FMCNTRNM 5-51
FMEXPNM 5-52
FMIDAT 5-53
FMIDNF 5-54
FMIDNT 5-55
FMNAME 5-56

TABLE OF CONTENTS (Continued)

Page

FMNLST 5-56
FMTDAT 5-57
FNDVAL 5-58
FREAD 5-59
FREEPG 5-60
FWRITE 5-61
GETBIT 5-62
GETDAT 5-63
The GETFILE Utility Package 5-64
  GETFILE 5-64
  GETLINE 5-68
  GETLINEB 5-69
  GETCLEAR 5-70
  GETPOS 5-70
  GETMARK 5-71
  GETSTART 5-71
  GETCLOSE 5-72
GETFLD 5-73
GETLABEL 5-74
GETPG 5-75
GETPGP 5-76
GETPPTD 5-77
GETPUNCH 5-78
GETRPTD 5-79
GETSTAT 5-80
GETTASK 5-80
GETTASKF 5-81
GLP 5-82
HEAPSORT 5-82
HEXBCD 5-83
HEXDMP 5-84
HEXLST 5-85
HEXNUM 5-86
HIBERN8 5-86
IDENT 5-87
IFIXD 5-87
INITCHAR 5-88
INITNM 5-88

TABLE OF CONTENTS (Continued)

Page

INSESORT 5-89
INSESRTN 5-90
INT 5-91
INTBCD 5-91
INTLST 5-92
IPDL 5-93
ISNGL 5-93
LABS 5-94
LBLGO 5-94
LBLSET 5-95
LDBYTF 5-96
LDBYTT 5-96
LDCHRF 5-97
LDCHRT 5-97
LFIX 5-98
LFIXD 5-98
LP 5-99
MAXIMUM 5-100
MINIMUM 5-100
MODIFILE 5-101
MOPEN 5-105
MOV 5-109
MOVB 5-109
MOVD 5-110
MOVEWORD 5-110
MOVF 5-111
MOVL 5-111
MVBYTF 5-112
MVBYTT 5-112
NXTCHAR 5-113
NXTFPG 5-114
NXTNAM 5-114
OCTBCD 5-115
OREAD 5-115
PAGFIL 5-116
PERROR 5-117
PNCHLDR 5-119
POP 5-119

TABLE OF CONTENTS (Continued)

Page

POPN 5-120
PPT 5-120
PPT1 5-121
PPTN 5-121
PUSH 5-122
PUSHN 5-122
PUTBIT 5-122
The PUTFILE Utility Package 5-123
  PUTFILE 5-123
  PUTLINE 5-124
  PUTLINEB 5-125
  PUTABORT 5-126
  PUTCLEAR 5-126
  PUTCLOSE 5-127
PUTFLD 5-128
PUTLABEL 5-129
The READBFIL Utility Package 5-130
  READBFIL 5-130
  READBYTE 5-134
  READBCLS 5-134
The READFILE Utility Package 5-135
  READFILE 5-135
  READBLOK 5-139
  READSECT 5-139
  READCLOS 5-140
  READMARK 5-140
  READPOS 5-141
  READTOP 5-141
RENAME 5-142
RESETLBL 5-146
RETERR 5-146
ROPEN 5-147
RPT 5-151
RPT1 5-151
RPTN 5-152
SELESORT 5-153
SETBIT 5-154
SETBREAK 5-154

TABLE OF CONTENTS (Continued)

Page

SETCOM 5-155
SETPG 5-156
SETPGP 5-157
SETPUNCH 5-158
SETSTRG 5-158
SHELSRTN 5-159
SNGL 5-160
SRTESTIO 5-160
SRWAITO 5-161
STBYTF 5-161
STBYTT 5-162
STCHRF 5-162
STCHRT 5-162
STYPEOK 5-163
STYPIN 5-164
SUBTRACE 5-165
TAPE 5-166
TAPENW 5-168
TESTIO 5-169
TESTTAPE 5-169
TIME 5-170
TOGGLE 5-170
The TRAVERSE Package 5-171
  TRAVERSE 5-172
  NEXTNODE 5-174
  ABORTRAV 5-175
TREAD 5-176
TREADNW 5-177
TSTBIT 5-178
TSTZERO 5-178
TWRITE 5-179
TWRITENW 5-180
TYPE 5-181
TYPEDBHX 5-182
TYPEDBI 5-182
TYPEHEX 5-183
TYPEINT 5-183
TYPEOK 5-184

TABLE OF CONTENTS (Continued)

Page

TYPIN 5-185
TYPOUT 5-185
UNATTACH 5-186
WAITIO 5-187
The WRITBFIL Utility Package 5-188
WRITBFIL 5-188
WRITBYTE 5-189
WRITBCLS 5-189
The WRITFILE Utility Package 5-190
WRITFILE 5-190
WRITBLOK 5-192
WRITCLOS 5-193
XEQTCOMM 5-194

Section Six
SYSTEM FORMATS 6-1

BIT STRING FORMAT 6-1
BYTE FORMAT 6-1
CHARACTER FORMAT 6-2
DATE AND TIME FORMATS 6-2
TEXT FILE FORMAT 6-4
Floating Point Format 6-4
Sign 6-4
Exponent 6-4
Mantissa 6-5
STORAGE FORMATS 6-5
Single Precision 6-5
Double Precision 6-6
Zero 6-6
Range 6-6
Accuracy 6-6
INTEGER FORMAT 6-7
MAGNETIC TAPE FORMAT 6-7
9-Track Tape 6-7
7-Track Tape 6-8

Index I-1

Section 1
INTRODUCTION

Section 1
INTRODUCTION

Prepared as a single source reference for the application programmer, this manual covers the system features most directly related to application programming. On-line documentation, FORTRAN considerations, file management, and system references are briefly described. Sections are arranged to correspond with the sequence of creating and executing application code. The loader, compiler, subroutine trace, and application debugger are presented in depth, along with system data formats, and system subroutines.

The Table of Contents and Index provide quick reference to material. Subroutines may be referenced by name, in the Table of Contents, or by function, in the Brief Subroutine Description section (Section 4). This publication is divided into six sections:

Section	*Title*	*Contents*
1	INTRODUCTION	Notation, access, levels, and commands for on-line documentation.
2	CREATING EXECUTABLE CODE	Step-by-step information on compiling and loading code; symbol tables, loader commands, and FORTRAN considerations are all included. There are sample files of loader commands, with directions for loading. Command execution and system references are also discussed.
3	DEBUGGING	An in-depth description of the Subroutine Trace tool and Application Debugger.
4	BRIEF SUBROUTINE DESCRIPTIONS	System subroutines, described and grouped by function.
5	SUBROUTINES	System subroutines for applications programming, presented in a detailed alphabetical list.
6	SYSTEM FORMATS	Data and storage formats.

Notation

This manual contains many examples of subroutine and command syntax. Differing type sizes and faces are used to distinguish between system and user responses.

- USER INPUT

Exact	*Variable*
ALL CAPS.	Upper and Lower Case.

- SYSTEM OUTPUT

Exact	*Variable*
ALL CAPS, SMALL TYPE FACE.	Upper and Lower Case Small Type Face.

- OTHER CONVENTIONS

↵	Carriage return.
n>	Operating System (O/S) level prompt for input.
[ ]	Square brackets contain optional material.
{ }	Braces contain two or more items, only one of which may be chosen.
x	A subscripted lowercase x at the end of a number enclosed in single quotation marks indicates a hexadecimal value (e.g., '8C5'x).

ON-LINE DOCUMENTATION

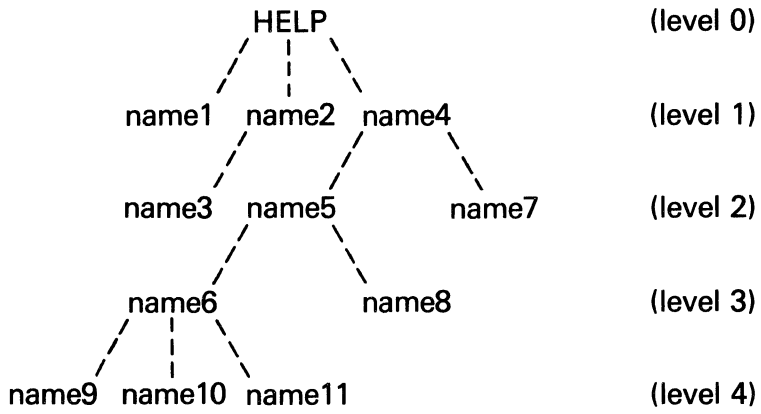
The following aspects of on-line documentation are described: access, levels, listing of documents, examples, and special commands.

Access

To access on-line documentation, type the HELP command in response to a prompt at the Operating System (O/S) level (n>). Documentation will be printed on the user COMDEV. While the HELP program is running, a ?] will ask for user input. A carriage return ↵ in response to the ?] exits HELP and returns user to O/S level.

Levels

HELP uses multiple levels of documentation to explain various features of the O/S:



Each name (name2, name4, name5, name6) may have a text document and further sub-category levels beneath it. Terminating names (name1, name3, name7, name8, name9, name10, name11) are documents without further sub-categories.

From any level, only documentation at lower levels can be referenced. To get this documentation, type the name of the document in response to the ?]. If the name typed has further sub-categories, you are moved to the next level. If the name is a document with no further sub-categories, you remain at the same level.

Listing Documents

From level 0 (HELP), the following commands will enable you to access the documents described.

Command	*Description*
name1	Name1 printed; you remain at level 0.
name2	Name2 printed; you go to level 1; the next level (name3) is listed for your choice of documents.
name4 name5	Name5 printed; you go to level 2; and documents at the next level (name6, name8)are listed.
name2 name3	Name3 printed; you remain at level 0 (name3 has no sub-categories).

Examples

These commands, in response to the level 0 HELP prompt ?], will access the documents described.

Command	*Description*
?] INFO FORTRAN	Information about FORTRAN/S is printed; you remain at level 0.
?] SUBROUTINE FILL	Information about the subroutine "FILL" is printed; you remain at level 0. For information on any system subroutine mentioned here, just type:

0> HELP SUBROUTINE Subroutinename ↵
In response to the O/S prompt.

Special Commands

Several commands make accessing documents more convenient. These commands are typed in response to the ?] prompt instead of a name:

/L	Lists the sub-categories and documents at the current level.
/B	Backs up one level and lists the sub-categories and documents at that level.
/R	Repeats the last text document printed.
/Q	Leaves HELP to return to the O/S.
/S and /A	HELP has two sets of documentation; system documentation and other (CADDS, user, etc.) documentation.
/S	Places you at level 0 of System documentation (the level of initial access to HELP).
/A	Accesses the alternate set of documentation, and puts you at level 0. This alternate set includes everything but System documentation. In order to return to System documentation, you must type /S.
HELP	Prints this document.
↵ (Carriage Return)	Returns to O/S level.

Section 2
CREATING EXECUTABLE CODE

Section 2
CREATING EXECUTABLE CODE

CREATING EXECUTABLE CODE

This is an overview of the steps involved in creating and executing application code on the CV Graphics Operating System (CGOS). Application code, as used in this manual, refers to any programs written to run in the graphics environment.

- Use the text editor to enter new routines onto the system (See the *CGOS 200 GNA Operator Guide* for text editor documentation). In general, an application program will consist of several routines. For a program stored as unbundled code, each routine should have a separate file. For bundled code, write the routine to the file CADDS.OVLY (Bundled and unbundled code are further described elsewhere in this section).

- Compile each new routine separately. One way to compile is to exit the editor with a File and Compile (FC) command. The compiler outputs a symbol map that is useful for debugging.

- Re-edit the file to eliminate any errors encountered during compilation. Repeat the editing/compilation process until each routine compiles without errors. You will now have a file of object code for each routine. These files are completely independent until loading resolves external references.

- To load a program, you must first create a text file containing loader commands (a Make file). Reference each file to be loaded by specifying the file containing its object code. Use standard load libraries to resolve references to system library routines.

- Enter the LOAD command from O/S level. Unless the NOTRACE option of the loader is used, the loader symbol tables will be saved; these tables are useful for debugging your program. The loader links each object module within the specified core area. At this point, an executable module exists.

- To execute application code on a trial basis, use the TEST command for bundled code or the RUN PROG command for unbundled code. In this example, CLD9001 is used as a sample coreload number. To execute bundled code (O/S level):

```
n>CADDS ↵
#SEL DEBUG DIR Dirname ADD Dirname//NLEV ↵
#TEST CLD9001 ↵
```

To execute unbundled code (O/S level):

```
n>CADDS ↵
#RUN PROG Filename ↵
```

FORTRAN/S

The special considerations deal primarily with overlays, and the arrays discussion deals with numbering and referencing.

Special Considerations

When user-written FORTRAN routines are included in an overlay created to run on CGOS 200 GNA, the overlay should also include the routines in the load library LOADLIB.FORTLIB. These routines support the execution of FORTRAN programs. LOADLIB.FORTLIB should be included in any Make which loads FORTRAN subroutines.

Arrays

In this manual, array numbering follows FORTRAN conventions. Indices begin at one (instead of zero as in TPL). To reference these arrays from TPL, just subtract one from the array index (i.e., OPTIONS(1) in FORTRAN is referenced as OPTIONS(0) in TPL).

Since arrays are referenced differently in TPL than in FORTRAN, use the system resident routines CMBYTT, LDBYTT, LDCHRT, MVBYTT, STBYTT, and STCHRT to manipulate bytes or characters from TPL programs. From FORTRAN programs, use CMBYTF, LDBYTF, LDCHRF, MVBYTF, STBYTF, and STCHRF.

COMPILATION

The method of compiling routines and the compiler symbol map are discussed.

Compiling Routines

SYNTAX

COMPILE filename [/OPTION1 $\left\{\right.$,OPTION2,...OPTIONn $\left.\right\}$]

PURPOSE

Compiles a TPL or FORTRAN source file.

OPTIONS

As follows (abbreviations in parentheses):

LIST	List source program and symbol map but not insert files.
LISTIN	Same as LIST but also lists insert files.
LO	List object code and symbol map.
MAPONLY (M)	List symbol map only.
NOFILE	Do not file object program.

CONDCOMP (C) Compiles source lines from the source program that begin with the (#) character. (Without CONDCOMP, lines that begin with (#) are not compiled.) CONDCOMP is typically used for debugging.

Note

The CONDCOMP option is treated differently in each language. In FORTRAN/S, the compiler replaces the (#) character with a blank. This makes it impossible to use a five-digit statement number with the CONDCOMP option. In TPL, the compiler removes the (#) character by shifting the entire source line one character to the left.

NEWCAT (N) Ordinarily, when a source or insert file is referenced, the compiler begins its search in the highest level catalog of the tree structure (the SYSCATLG). NEWCAT designates a catalog that will be searched before the SYSCATLG.

Example

If NEWCAT = T is specified and the compiler needs a file to read, it prefixes "T." to the filename and searches for that file. The file will be used if it exists. Otherwise the compiler will try to access the file originally named (no T prefix). If neither file exists, the compiler gives an error and halts.

Note

When a user lacks access to an object file and NEWCAT is specified, the NEWCAT name will be prefixed to the object filename.

DECL — Default setting that lists all undeclared symbolic names. A symbolic name is undeclared if it is not a subroutine name or a statement label and is not defined in any of the following kinds of statements:

FORTRAN/S	*TPL*	
SUBROUTINE	E:F	D:T
FUNCTION	E:O	B:D
COMMON	G:L	V:S
DIMENSION	D:N	SUBR
INTEGER	I:R	ENTR
REAL	F:T	EQU
DOUBLE INTEGER	D:R	TEQ
DOUBLE PRECISION		
EXTERNAL		
DATA		

FUNC — Lists all undeclared symbolic names in the source program (including external functions and subroutines).

USED — Default setting for listing any unused symbolic names that do not appear in executable code or as TPL symbolic constants.

Note

DECL and USED are debugging aids for uncovering typographical errors that might otherwise go unnoticed. The compiler defaults to these two options.

BRIEF (B) — Minimizes compiler output messages by suppressing messages generated by the DECL and USED options

Symbol Map

The compiler outputs a symbol map as part of the compilation process. Where applicable, the symbol map lists a location, a mode, a reference type, a dimension, and an equivalence for symbolic names used in the source program or insert file. These are output to the COMDEV in columns, under the headings LOC, MOD, REF, DIM, and EQUIV.

Table 2-1
Elements of a Symbol Map

Heading	Description
LOC	Location is based on six different characteristics and depending on the type of symbolic name. Each is listed and described briefly below: • Local variables — the offset from 0 in the low data area. • Global variables and external subroutines (External Symbols) — The value is a two-digit hex index assigned by the compiler, which assumes that External Symbol is defined somewhere outside the source program. During compilation, the first External Symbol is assigned the value of 'FF'x, the second is assigned 'FE'x, the third 'FD'x, and so on. The loader uses the index values to resolve all references to External Symbols. • TPL statement labels, internal subroutines, or subroutine entry point names — the offset from the start of the code. • FORTRAN equivalenced variables — the offset from the start of the base symbol listed under EQUIV. • TPL based variables — pointer location is listed. • TPL symbolic constants — lists value of constant instead of its location.
MOD	Abbreviations, in the column under MOD and the row of a symbolic name, are as follows: I Integer variable. F Floating point variable. I2 Double integer (32-bit) variable. F2 Double precision floating point variable.

Table 2-1
Elements of a Symbol Map (Continued)

Heading	Description
REF	Abbreviations under REF refer to the following types of symbolic names: L Local variable. A Dummy argument. B TPL based variable. G Global variable. XS External subroutine. SE Subroutine entry point. FE FORTRAN function entry point. IS Internal subroutine. IF FORTRAN internal function. LB TPL statement label. C TPL symbolic constant.
DIM	Refers to the dimension of a variable, where applicable: • Array variables — DIM is the number of elements in the array. For a TPL array dimensioned with an asterisk(*), however, DIM is −1. DIM will be one less for a TPL array than for a FORTRAN array. EXAMPLE (For an array of 19 elements): *FORTRAN*: DIM ARRAY 19 *TPL*: DIM ARRAY 18

Table 2-1
Elements of a Symbol Map (Continued)

Heading	Description
	• Scalar variables — there is no dimension. • FORTRAN COMMON block — the total number of words in the block.
EQUIV	Base symbol used in calculating the location of a FORTRAN equivalenced variable.

LOADING CODE

Object code is linked and loaded according to commands in a text file. The CGOS loader processes this text file (called a Make file) to tie together each file of code and all external references.

The O/S command LOAD activates the linking loader. LOAD references a Make file containing loader commands that designate:

- Files containing subroutines and functions to be loaded.
- The overlay into which object code is loaded.
- Linkage for system references in the object code.
- Global areas.

These commands map out an area of core along with the code and data to be put in this area.

When LOAD is successfully executed, object files are linked together to form an overlay (coreload) stored on disc. The overlay is composed of binary code in an executable form.

Creating the Make File

The following general rules apply in the creation of a Make file:

- Lines beginning with an asterisk(*) in column 1 are ignored by the loader as comment lines.
- No blank lines are allowed.
- FORTRAN common blocks are indicated by a C& and the common block name.

- All values (Val1,Val2,etc.) in the Make file are hexadecimal numbers.
- Prefix a zero to hexadecimal values with a first significant digit of A-F (i.e., F57C must appear as 0F57C).
- Alphabetic values are in "quotes". They are always packed two to a word and are left-justified.

Loader Keywords

The following loader keywords are valid commands for the Make file:

BLOCK

BLOCK Val1,Val2

Declares a region of core allotted to code and data from loader commands that follow:

- Val1 Lower bound of the block (inclusive).
- Val2 Upper bound of the block (exclusive).

BLOCK causes an END-OF-BLOCK condition on the previous block.

Under an end-of-block condition, the loader compares the amount of core left in block to the block size. One of two messages is printed:

XXXX WORDS AVAILABLE IN BLOCK

denotes adequate space. If inadequate space remains for code and data, the message:

OVERFLOWED BLOCK BY XXXX WORDS

will be printed as soon as the overflow is detected. When a block has overflowed during loading, WRITE, OVWRITE, CWRITE, and EXECUTE commands are not processed.

CORORG

CORORG Val1 [,Val2]

Allocates internal buffer space to the overlay created by the Make. Val1 is the lowest address available to the Make. That is, no BLOCK command may define a block that starts below Val1. Val2 may be omitted and will default to the value 8000 — Valix. Val1 + Val2 is the highest address that my be set by the Make. That is, no BLOCK command may define a flock that ends above Val1 + Val2.

CWRITE

CWRITE Val1, Val2, Val3

Writes the core image to disc and causes an end-of-block condition. After completing the disc write, the loader exits to O/S level. The core image is stored in the file named in the FILENAME statement. At O/S level, this file can be executed with the RUN command. CWRITE accepts the following arguments:

Val1 Number of words in the overlay. (This is the number of words that will be written out to the disc.)

Val2 Starting core location of the overlay. When the overlay is run, this will be the starting address of the executable code.

Val3 Highest core address used when the overlay is executed. The loader will check to ensure that there is sufficient memory available for the task to execute the overlay.

DUMP

DUMP Val1, Val2

Prints overlay locations from address Val1 to address Val2 as a standard hex dump.

END

Signifies the end of a text file. The loader interprets it in either of two ways:

- As the end of an insert file.
- As a command to return to O/S level without filing the Make overlay to disc.

ENDC

Marks the termination of a group of conditional loader commands. Conditional commands must begin with an IF. It is illegal to have an ENDC command without a matching IF command.

ELSE

A conditional statement used after an IF to enable execution of code when the IF condition is false. ELSE appears before the ENDC statement (see example under IF).

EQU

EQU Sym Val[,Val2]

Defines symbols; the symbol SYM is assigned the value Val1 unless Sym is already defined (see also REDEF and UNDEF. If Val2 is used, Val1 is assumed to be a subroutine address and Val2 is the number of arguments required by the subroutine.

FILENAME

FILENAME Filename

Specifies the overlay file to be used by WRITE, OVWRITE, or CWRITE. This overlay file may be overridden by another FILENAME command, or by the PRIMARY, OVCAT, or DEBUG loader options. If FILENAME is omitted, the overlay filename will default to the Make filename.

FILL

FILL Symbol = Value

Sets initial values for an array of memory locations within the overlay being loaded. The array defined by Symbol will be set to Value for each word assigned to the array by a GLOBAL statement.

IF

IF Condition

In conjunction with ENDC, IF controls processing of other loader commands. Commands between the IF and ENDC delimiters are processed only if Condition is true. If Condition is false, no subsequent loader commands will be processed until a matching ENDC or an ELSE is found. CONDITION may be any combination of BOOLEAN EXPRESSIONS and OPERATORS.

BOOLEAN Expressions	*FORTRAN Equivalents*
Val1 = Val2	TRUE IF Val1 .EQ. Val2
Val1 # Val2	TRUE IF Val1 .NE. Val2
Val1 < Val2	TRUE IF Val1 .LT. Val2
Val1 > Val2	TRUE IF Val1 .GT. Val2
Val1 ≤ Val2	TRUE IF Val1 .LE. Val2
Val1 ≤ Val2	TRUE IF Val1 .GE. Val2

OPERATORS:

Several operators are available for loader Make files. They are classified as Integer or Boolean depending on the type of value they return. Boolean operators are AND, NOT, OR, UNDEFP, and UNUSEDP. Integer operators are SIZE, LAND, and MAX.

The syntax for operators is:

[Operator Arg1, Arg2,...Argn]

Square brackets and the first argument are mandatory. Other arguments may be optional, depending on the operator. Use commas with multiple arguments.

- Boolean

  AND — True when each of the following arguments is true, otherwise false.

  NOT — Takes one argument and returns its logical negation.

  OR — True when any of the following arguments is true, otherwise false.

  UNDEFP — Takes one symbol name and returns true if the symbol has been defined and false otherwise.

  UNUSEDP — Takes one symbol name and returns true if the symbol has been used and false otherwise.

- Integer

  LAND — Takes an arbitrary number of integer arguments and does a bitwise AND, returning the result.

  MAX — Returns the largest of an arbitrary number of integer arguments. Two's complement signed comparisons are used.

  SIZE — Returns the number of words of storage occupied by one symbol.

  EXAMPLE:

```
IF [OR [UNDEFP Label1],[SIZE Label2] = 2]
   .
   .
   .
ELSE
   .
   .
   .
ENDC
```

INSERT

INSERT Filename

Designates an additional file to include in the Make. The loader stops processing the original Make file to process the entire insert file of loader commands. When an END command is encountered in the insert file, the loader resumes processing the original Make file.

GLOBAL

GLOBAL Sym1 (Val1), . . .,Symn (Valn)

Defines and allocates global variables. Sym is the name of each global variable and the corresponding Val is the length in bytes (dimension) of the block of core to allocate for Sym. Val must appear; even a value of 1 must be stated explicity (see the SEP and NOSEP commands).

Note

Use a value of zero to allocate a global of zero words. For example: GLOBAL X(0) is the same as EQU X for the last core location used for a global, or for code allocation.

LIB

LIB Filename

Searches a load library (specified by Filename) for unresolved subroutine references. Only unresolved references will be loaded from the library.

LISTSYM

Default setting for listing symbols in a table as they are created. LISTSYM and NLISTSYM drive a counter that is initially set to zero. Whenever symbols are created and the counter is less than one, those symbols are listed in the table. When the counter is greater than or equal to one, symbols are not listed. NLISTSYM adds one to the counter; LISTSYM subtracts one from the counter.

LOAD

LOAD Catalog/File1, . . .,Filen

Loads object code from a list of object files in the same catalog. The resulting absolute binary code is placed into the current block. The name of the file and the addresses of the code and data sections from that file are added to the listing in this format:

Catalog.File1 XXXX XXXX XXXX

(See the SEP and NOSEP commands.)

LOADLIB

LOADLIB Filename/Subr1,Subr2....

Loads specific routines from the load library specified by Filename.

LOADSYM

LOADSYM Filename

Replaces the current loader symbol table with the symbol table saved in the file specified with LOADSYM. A loader symbol table file can only be created with SAVESYM. LOADSYM should be used at the start of the Make, before any other symbols are created.

NLISTSYM

Inhibits the symbol table listing (See LISTSYM).

NOSEP

Invokes NOSEP mode of loading (SEP mode is the default setting). In NOSEP mode, all code and data is loaded at the start of the block (low addresses). The following loading rules apply:

LOAD	Code section at start of block. Globals are immediately after the code section. The data section is immediately after the globals.
GLOBAL	Globals at start of block.
TV	TVs at start of block.

OVWRITE

OVWRITE Val1,Val2,Val3

Writes the overlay to disc. Instead of exiting to O/S level after completing the disc write, the loader will continue to process the Make file.

Val1	DLOC within the file for writing the overlay.
Val2	Number of words in the overlay. (This is the number of words that will be written out to the disc.)
Val3	Starting core location of the overlay.

PRINT

PRINT Text string

Adds one-line messages to the output. Any text following the PRINT keyword will be output literally to the COMDEV when the routine is loaded. This message cannot be inhibited with any setting of the loader LIST option.

REDEF

REDEF Sym Val

Defines symbols. Unlike EQU, it does not discriminate against previously-defined symbols. REDEF Sym Val is equivalent to:

UNDEF Sym followed by EQU Sym Val.

SAVESYM

SAVESYM Filename

Saves the current loader symbol table in a file that can be referenced and re-used with LOADSYM.

Most Makes begin with one or more "standard" insert files. Those insert files often define symbols that are common to these "standard" Makes. Rather than process the insert files as the first part of every Make, it is more efficient to process the insert files and save the resulting symbol table in a file. That file can then be used by every Make that would otherwise use the insert files to re-create the symbol table.

SEP

Invokes the SEP mode of loading (SEP mode is the default). In SEP mode, code is loaded at the start of the block (low addresses); data is loaded at the end of the block (high addresses). The following rules apply for loading in SEP mode:

LOAD	Code section at start of block. Globals at end of block. The data section is at the end of the block (before the globals).
GLOBAL	Globals at end of block.
TV	TV's are at the start of the block.

SET

SET Sym = Val1,. . .,Valn

or

SET Sym (Offset) = Val1,. . .,Valn

Assigns memory locations within the overlay to values (Val1,..Valn). Sym is a global, and up to 15 values can be specified. Val1..Valn are assigned to memory locations beginning at the position of Sym plus Offset. As in TPL indexing, Offsets start at zero for the first word of the global (Offset defaults to zero).

TV

TV Sym1, . . .,Symn

Allocates transfer vectors (see Loader Glossary). The TV command is unusual in that it allows references to symbols that are not yet defined. One two-word TV is allocated for each symbol in the list.

TVs are generally used to reserve entry points to overlay routines, and are therefore defined immediately following a BLOCK command. The TV command allows the programmer to reference a routine without knowing in advance where it will be loaded into memory.

UNDEF

UNDEF Sym1,. . .,Symn

Ensures that symbols in the argument list are undefined. Any undefined symbols are created, and existing symbols lose their definitions so that they may be redefined.

WRITE

WRITE Val1,Val2,Val3,Val4,Val5

Writes the overlay to disc and causes an end-of-block condition. On completion of the disc write, the loader exits to O/S level. The core image is stored in the file named in the FILENAME statement. At O/S level, this file can be executed with the RUN command. WRITE accepts the following arguments:

Val1	The DLOC within the file that the overlay will be written out to.
Val2	(Not used)
Val3	(Not used)
Val4	Number of words in the overlay. (This is the number of words that will be written out to the disc.)
Val5	Starting core location of the overlay.

Running the Loader

To activate the loader, use the O/S command LOAD.

SYNTAX

LOAD [Make file[/OPTIONS]]

Make file — Text file containing loader commands.

To enter loader commands from the COMDEV instead of a MAKE file, just type: LOAD) and then type them in.

OPTIONS

Options must be preceded by a slash. When more than one option is entered, each option must be separated by a comma. OPTIONS are as follows:

NEWCAT Format:

NEWCAT = Catname

or

N = Catname

Ordinarily, when a file is referenced, the loader begins its search in the SYSCATLG. This option specifies the name of a catalog to search before SYSCATLG.

EXAMPLE:

LOAD Makefile/NEWCAT = Catname

- Reading

  The statements:

  LOAD Filename.
  INSERT Filename.
  LIB Filename.
  LOADLIB Filename.
  LOADSYM Filename.

  cause the loader to read a file. With NEWCAT = Catname, the loader will search for the file: Catname.Filename. If that file does not exist, the loader will try to read the file Filename. If neither file exists, the loader gives an error and returns to the O/S.

- Writing

  The loader writes to a file when it encounters one of the following statements:

  WRITE Filename.
  CWRITE Filename.
  OVWRITE Filename.
  SAVESYM Filename.

If NEWCAT = Catname is specified and the file Catname.Filename exists, it will be overwritten by a new file. Otherwise, the file Filename is created and written.

OVCAT — Format:

OVCAT = Catname

Activates a special mode of loading, used for debugging CADDS. Filenames specified within the Make are ignored. The overlay is always written to the file named "Catname.Make filename" at disc address zero. The CADDS overlay manager is equipped to handle overlays written using the OVCAT option.

DEBUG — Format:

DEBUG = Catname

DEBUG is a combination of the NEWCAT and OVCAT options (i.e., DEBUG = T is equivalent to NEWCAT = T, OVCAT = T).

RESTRICTION: Only one of these options (NEWCAT, OVCAT or DEBUG) may be specified.

NOFILE — Loads and tests an overlay without writing it to disc. The message:

```
**TEST RUN**
```

is printed when loading is completed.

NOTRACE — Disables saving of the loader symbol file. Symbolic entry point names will not be available for access by the DEBUG and TRACE commands. (For Trace and Debug information, see Section 3.)

LIST — Activates listing of the Make file and loader symbol map — useful for finding typing mistakes in the Make file.

BRIEF — Suppresses listing of available block space, along with output from any DUMP keywords in the Make file.

LISTIN	Turns on listing of the Make file, loader symbol map, and all insert files.
MAPONLY	Turns on listing of the loader symbol map.

Loader Glossary

CORE IMAGE	A stream of bits, whose "final" destination is in memory. A core image is usually executed as machine instructions.
CORELOAD	An overlay containing a module of executable graphics code, a coreload is the result of successful execution of the LOAD command.
GLOBAL	A variable available to more than one subroutine.
INSERT FILE	A text file containing loader commands; the Insert file is included in a Make using the loader command INSERT.
MAKE FILE	A text file containing loader commands
OBJECT FILE	A binary output file from a compiler that will be processed by the loader is called an object file or object program.
OVERLAY	A file containing a core image
SOURCE	A program or subroutine that is in the form of a text file is called a source file or a source program (see OBJECT).
TV	An abbreviation for transfer vector. A TV uses two words of memory to create the assembly language instruction for a GOTO. The first word is a "JMP @.+1". The second word is a core address. A subroutine call to the location of the TV is equivalent to a subroutine call to the address in the second word of the TV.

Loader Errors

Loader error messages are contained in three files listed below. These files divide the errors into three categories. Diagnostic errors are the least serious, Minor errors are more serious, and Major errors cause the Loader to abort processing of the current Make and return to O/S level. Please refer to on-line documentation if there is any question regarding codes and messages.

Message File	*Error Range*
SYSNEWS.ERROR.LOADER.MAJOR	(8001-8052)
SYSNEWS.ERROR.LOADER.MINOR	(8080-9092)
SYSNEWS.ERROR.LOADER.DIAGNOSTICS	(80D1-80D5)

The following list contains error numbers and associated error messages. All error numbers are hexadecimal values.

- Loader Diagnostic Errors

  80D2 An external reference fill-in chain starts at location zero. This occurs when an external reference must be loaded into a block starting at 0000.

  80D4 Object file references a GLOBAL that was not defined as a GLOBAL. A GLOBAL must be defined using a Loader GLOBAL command or allocated when an object file is loaded.

  80D5 A subroutine call or definition has the wrong number of arguments. There are three frequent causes:

  - The subroutine was previously defined and a call uses a different number of arguments.
  - The subroutine was previously called and a call uses a different number of arguments.
  - The subroutine was previously called and a definition specifies a different number of arguments.

- Loader Minor Errors

  8080 Attempt to change the mode of a symbol.

  8081 Attempt to set ESI to a value greater than 'FF'.

  8086 Object file entry point is already defined.

  8088 Format error in object file: last word is not the end-of-file flag ('FFFF').

  808A Internal checksum error in object file.

  808B External checksum error in object file.

  808C the object file being loaded is referring to a global that is smaller than the declaration in the source file.

  808D Bad character where a slash is expected in a LOAD or LOADLIB command.

  808E Load library file is not a loader data file.

  808F Load library file is a loader data file but does not contain a load library.

  8090 Load library file is in an old format that cannot be used by the current version of the loader.

  8091 File to be loaded is not a TYPE 2 (object) file.

  8092 Extra characters at the end of a Loader command line.

- Loader Major Errors

8001 Attempt to read the size of a symbol that is not a global.

8002 Attempt to execute when a block has overflowed.

8005 Attempt to store a word at an address outside of internal buffer space allocated in the CORORG command. This error is due to an attempt to load a program or initialize a global at a bad address.

8006 Attempt to read a word from an illegal address — this indicates that the current block or an earlier block has overflowed.

800D Format error in object file — data block has a bad count.

800E Set location block in object file trying to set location at a symbol that is not yet defined. This error occurs when TPL source code tries to data initialize (vector values) a global of indefinite length (dimensioned with a *) and the location of that global is not yet defined.

8010 Internal error in object file.

8011 Bad character where a slash (/) is expected in the Loader (O/S) command line.

8012 CORORG command specifies a block outside the range 0000-8000.

8013 INSERT command in an insert file (it is illegal to use nested INSERTs).

8015 Bad character where an equal sign (=) is expected in a FILL command.

8016 Symbol in a global command is already defined.

8017 Bad character where a parenthesis [) or (] is expected in a global command.

8018 Bad character where a comma (,) is expected in any of the following commands: GLOBAL, SYM, TV, or UNDEF.

8019 Illegal function.

801A Attempt to execute when a symbol is not defined.

801B Bad character where a right square bracket (]) is expected in a function expression.

801C Symbol specified in a SET command has an undefined location.

801D Bad character where a right parenthesis [)] is expected in a SET command.

801E Bad character where an equal sign (=) is expected in a SET command.

801F Attempt to define a symbol that already has a definition using an EQU command.

8021 Bad loader command: keyword not recognized.

8022 Illegal option on LOADER (O/S) command line.

8023 Overlay file will be bigger than 'FFFF' sectors.

8024 Internal loader error, attempt to define a symbol that is already defined.

8028 Symbol table overflow: too many symbols have been created.

8029 XRWAD table overflow.

802A Too many total entry points, called subroutines and globals referenced from an object file.

802B Attempt to execute with low end of CORORG buffer below 2000x or high end above 8000x. Low end of CORORG buffer is the address following the 'CORORG' keyword in the Make file. The high end defaults to.

8000x unless specified in the CORORG command line.

802C Start address specified in an execute command has an address outside the overlay that was created.

802D NEWCAT was specified in the Loader command line, but the overlay file cannot be modified because it is in use.

802E Block to be written to disc overlaps the overlay as defined in CORORG command.

802F Left parenthesis [(] in an expression does not have a matching right parenthesis [)].

8030 Symbol referenced in an expression is not defined.

8031 Two operators or two operands in a row in an expression.

8032 Null strings are illegal.

8033 Command does not have enough numerical arguments.

8034 Command has too many numerical arguments.

8035 First operand of a Boolean expression is a string longer than two characters.

8036 Illegal Boolean operator.

8037 Second operand of a Boolean expression is a string longer than two characters.

803B Entry point not found in load library.

803C Internal loader error.

803E Attempt to data-initialize a global using a SET command exceeds the limits of the global.

803F ESI jump instruction address falls outside of address range (−128 to +27 words).

8040 Low data section of object file has exceeded 100x. The loader will expand the low data section of an object file in order to create linkages to other subroutines and to globals. The resulting size after expansion cannot be larger than 100x. This error condition can usually be satisfied by moving local integer arrays into a common area (FORTRAN) or converting them to globals (TPL), then recompiling the object file and reloading.

8042 Overlay filename not specified.

8043 Overlay file is larger than 'FFFF' sectors long.

8044 An object file created from a TPL source file has referred to a global of indefinite length (using a * declaration), but the global is undefined when the object file is loaded.

8045 Attempt to save a symbol table containing an undefined symbol.

8046 Filename not specified in LOADSYM or SAVESYM command.

8047 File specified for a LOADSYM command is not a loader data file.

8048 File specified for a LOADSYM command is not a saved symbol table.

8049 Symbol table in the file specified for a LOADSYM command is in an old format and cannot be used by the current version of the loader.

804A File specified for a LOADSYM command has a bad checksum.

804B Arguments to BLOCK command are in the wrong order — the starting address of the block is larger than the ending address of the block.

804C No more lines in input file — this can happen when there is no END command to terminate an insert file, or when a Make file is not terminated by a CWRITE, END, EXECUTE, or WRITE command.

804E BLOCK command specifies a block outside of addressable memory.

804F Attempt to allocate a negative number of words — GLOBAL command may have specified a bad global size.

8050 FILL statement is data initializing a symbol that is not a GLOBAL.

8051 SET statement is data initializing a symbol that is not a GLOBAL.

8052 Function value is not an integer.

8083 Object file is incompatible with the current version of the loader because it was compiled with an old version of the compiler.

LOADLIB

COMMAND — LOADLIB

PURPOSE — Creates a load library.

SYNTAX — LOADLIB Filename [/options]

COMMENTS — A load library is a collection of object files merged into one and indexed by entry point name. There are three reasons for grouping object files in this way:

- Speed — using the LOADLIB command in a Make, object files can be loaded directly from the load library. This speeds up the loading process by reducing access to the file manager.

- Ease of use — in a Make, the LIB command accesses the load library created with LOADLIB. LIB is also used to access the standard load libraries:

  CADDSLIB
  FORTLIB
  OSLIB

  These standard libraries should always be referenced in this order, otherwise some subroutine references may remain undefined. LIB directs the loader to resolve undefined subroutine references by loading necessary object files from a load library.

- One file replaces many files — packing many small files into one big file reduces the amount of disc space needed for libraries of object files. It also reduces the time required to save (or restore) those object files on tape.

FILENAME — Before calling LOADLIB, create a text file containing the LOADLIB commands:

```
ADD Catalog/Filename1 [,Filename2. . .]
```

Adds the object files Catalog.Filename1, Catalog.Filename2 (etc.) to the new load library.

```
ADDCAT Catalog
```

Adds the entire catalog to the load library.

FILENAME Filename

Names the load library file. In the Make, Filename is the name used to reference the load library. Use the FILENAME command exactly once in each load library. If there is more than one FILENAME command, the last one wins. The default filename is the name of the LOADLIB text file.

OPTIONS LIST [= Hexnum]

Controls the type of listing produced by LOADLIB. The hex number is a bit mask; each bit designates items to be added to the listing.

Mask	*Meaning, List:*
1	Input filename.
2	Input text file (first pass).
4	Object filenames (first pass).
8	Entry point names (first pass).
10	Object filenames (second pass).
20	Entry point names (second pass).
40	Load library filename.

If list is specified without a value:

LOADLIB Filename/LIST

LOADLIB defaults to listing the Input filename and the Load library filename.

LISTLOAD

COMMAND LISTLOAD

PURPOSE Searches a text file for commands that reference other files. From these command references, a list of files is created and output to a file or the COMDEV. Optionally, LISTLOAD can generate a system command for each file referenced.

COMMENTS LISTLOAD searches a text file for keywords. These keywords are loader (Make) commands that reference other files. Generally the text file is a MAKE file, but LISTLOAD can also search a source file for the /INCLUDE compiler command. Optional arguments control which commands will be searched for. LOAD, INSERT, LOADSYM, and LIB are all commands that tell the loader to access a file or set of files. They are also arguments to LISTLOAD.

SYNTAX

LISTLOAD Output filename/KEYWORDS[,OPTIONS]

KEYWORDS Separate multiple keywords with a comma. There are no default keywords. Keywords are:

LOAD	Include files referenced in LOAD commands.
INSERT	Include files referenced in INSERT commands.
LOADSYM	Include files referenced in LOADSYM commands.
LIB	Include files referenced in LIB commands.
LILL	Process all types of files specified in the above options.
INCLUDE	INCLUDE files from source code referenced by /INCLUDE statements.

OPTIONS Control the output of LISTLOAD:

COMDEV	Output list of files to COMDEV, may not be used as a single option or with GENCOM.
GENCOM	Generate a system command for each file referenced, may not be used as a single option or with COMDEV.

OUTPUT

LISTLOAD will respond to the initial command line with a prompt:

```
#
```

This prompt indicates that the user should type a catalog name.

EXAMPLE:

```
1> LISTLOAD CAB.TEST.LIST/COMDEV,LOAD INCLUDE
# CAB.TEST
```

- COMDEV

  When the COMDEV option is declared, LISTLOAD prints out names of appropriate files in the catalog.

EXAMPLE:

```
1> LISTLOAD LL.TEST.LIST/GENCOM,LOAD
   ENTER COMMAND SKELETON
   PRINTLP $$ (↓)
#> MAKE.COMMAND.LOAD ↓
#  ↓
1>
```

- GENCOM

  In response to the GENCOM option, LISTLOAD prints:

  ```
  ENTER COMMAND SKELETON
  ```

  Enter the desired command followed by at least one space and one or two pairs of dollar signs. If two pairs are used, commands will be generated in the following format:

  ```
  MOVE $$, CAB.$$
  ```

  After a carriage return, LISTLOAD prompts the user for a catalog name.

  Once the preceding example is executed, the first few lines of LL.TEST will look like:

  ```
  PRINTLP COMMAND.LOAD.ALLOC
  PRINTLP COMMAND.LOAD.DIAGERR
  PRINTLP COMMAND.LOAD.DOINDEX
  PRINTLP COMMAND.LOAD.GETADDR
      .       .       .       .
      .       .       .       .
      .       .       .       .
  ```

  In the output file, LOADLIST replaces the GENCOM command "$$" with each filename.

- Mulltiple Catalogs

  When LOADLIST is finished processing the original catalog, it will prompt for another catalog with:

  ```
  #
  ```

  A carriage return, will invoke the output phase of LISTLOAD. When output is complete, LISTLOAD will return to O/S level.

GENINDX

PURPOSE

Scans a source file or catalog to generate an index of entry points, globals, or subroutines.

SYNTAX

GENINDX {ENTPNTS,GLOBALS,SUBRS} [/OPTIONS]

ENTPNTS	Files associated with each entry point.
GLOBALS	GLOBALS and files that reference them.
SUBRS	Subroutines referenced by each file.

OPTIONS

One or more of the following entries, separated by commas:

NOSORT	After generating file, do not sort generated entries.
PRINT	Print each line as it is added to the file.
NOFILE	Do not generate an output file. No operation will be performed if NOFILE is chosen without PRINT.
UTIL	Effective only with index type SUBRS. Include subroutines that have the final level of names begin with &. (This includes compiler-generated calls to run-time library in file).
MRGCOPY	If SORT pass generates multiple lines with identical sort fields, blank out sortfield in all except the first line.
PSTAT	Print status information during sort pass, if sort pass is included.
SINGLE	SORT pass suppresses listing of multiple entries with identical sort fields.
MULTIPLE	Only SORT pass generated multiple entries with identical sort fields are included in output listing.

INPUT

After scanning the command line, the system will prompt for the output filename (unless NOFILE is chosen). Next, it will prompt for the name of a catalog in the file manager system (SYSCATLG is a valid choice).

OUTPUT

GENINDX will search the catalog to generate the type of index specified in the command line. After processing the catalog, GENINDX will continue to prompt for additional catalogs until the user responds to the (#) prompt with a carriage return.

General format of GENINDEX dialog.

First, the command line is echoed on the COMDEV, along with time and date:

```
GENINDX Indextype/Options Time Date
```

Next, you are prompted for the output filename and the catalog to be searched:

```
OUTPUT FILENAME: Filename
# Catalog
```

COMMENTS

- If the SORT option is chosen, elaspsed time may be great (on the order of hours) for a very long index.
- With the PSTAT option, status information is printed at regular intervals, informing user that system is still running.

EXAMPLE

Sample GENINDX dialog:

```
n> GENINDX SUBRS/UTIL ⏎
OUTPUT FILE NAME:HELP.SUBRS ⏎
#SYSCATLG ⏎
#
n>
```

CROSSREF

PURPOSE

Lists object file entry points and external references.

SYNTAX

```
>CROSSREF ⏎
# FILESPEC
```

INPUT

FILESPEC Indicates object files to be cross-referenced:

Catalog Name/File1,File2,xxxFilen

To cross-reference various files in one catalog.

Catalog Name//NLEV or Catalog Name

To cross-reference the entire catalog of object files.

OUTPUT

For each file to be cross-referenced, CROSSREF Lists subroutines called, entry points and globals. CROSSREF will continue to prompt for additional catalogs until a carriage return is entered in response to the prompt.

EXAMPLE

Sample CROSSREF dialog:

```
n> CROSSREF
# SYS.FM/FMSETDN
**************************************

  CODE SIZE: 0040       DATA SIZE: 000A

  ENTRY POINTS:
    FMSETDN

  GLOBALS:
    TSKODATA TSKOLIST TSKERR

  SUBROUTINES:
    &SETUP &SETDWN MOV

#
n>
```

SYSTEM REFERENCES

System references may be either symbolic or explicit.

Symbolic References

References to Operating System (O/S) globals and subroutines may be defined with the following commands for referencing loader INSERTs:

For O/S level commands:

```
INSERT JANMAKE.SYM.BASIC      (disc and memory areas)
INSERT JANMAKE.SYM.SYS        (resident system references)
INSERT JANMAKE.SYM.SYSOV      (system overlay references)
```

And, if desired:

```
INSERT JANMAKE.SYM.DBLINT     (long integer utilities)
INSERT JANMAKE.SYM.FLPT       (floating point utilities)
INSERT JANMAKE.SYM.FMSYS      (file management references)
```

Note

SYMFILEs will define these references more efficiently. There is also a standard SYMFILE for defining CADDS references.

Explicit References

Avoid using explicitly defined references in a loader source file. Numeric values and offsets are subject to change with each O/S revision. A loader source file with explicit references may need modification to run on a revised O/S. To avoid this problem, use the symbolic references in the preceding INSERTs or the SYMFILEs listed below.

Symfiles

Symfiles define O/S references and CADDS references.

O/S REFERENCES

References to O/S globals and subroutines may be defined with the following SYMFILEs. These SYMFILEs contain all the references from one or more system INSERT files. Only one SYMFILE may be used, and it must appear in a LOADSYM statement before any loader source file references are defined. The standard SYMFILEs and their contents are:

SYMFILE	*INSERT File References*
JANMAKE.SYMFILE.SYSFM	BASIC + SYS + FMSYS
JANMAKE.SYMFILE.SYSFMLI	BASIC + SYS + FMSYS + DBLINT
JANMAKE.SYMFILE.SYSOV	BASIC + SYS + SYSOV
JANMAKE.SYMFILE.SYSOVFM	BASIC + SYS + SYSOV + FMSYS
JANMAKE.SYMFILE.SYSOVFMLI	BASIC + SYS + SYSOV + FMSYS + DBLINT
JANMAKE.SYMFILE.SYSOVFP	BASIC + SYS + SYSOV + FLPT
JANMAKE.SYMFILE.SYSOVFMLIFP	BASIC + SYS + SYSOV + FLPT + FMSYS + DBLINT
JANMAKE.SYMFILE.SYSOVFMFP	BASIC + SYS + SYSOV + FMSYS + FLPT

To load system library routines, include LIB statements for LOADLIB.OSLIB and LOADLIB.FORTLIB in your loader source file.

CADDS REFERENCES

There is a standard SYMFILE for defining CADDs references:

INSERT.SYMFILE.STANDARD

EXECUTING APPLICATION CODE

Most application code is written to be executed within CADDS; this code is divided into two categories; bundled and unbundled. Bundled code and unbundled code differ in storage location and mode of access.

- Bundled

  All bundled code is written to one disc file — CADDS.OVLY. CADDS.OVLY is made up of fixed-length blocks (overlays). Each overlay, created when a Make is processed, is written to a different offset in CADDS.OVLY. These overlays are referenced by "coreload numbers" that address a portion of the CADDS.OVLY file. Even within CADDS.OVLY, one overlay has no relationship to another except through references within code.

- Unbundled

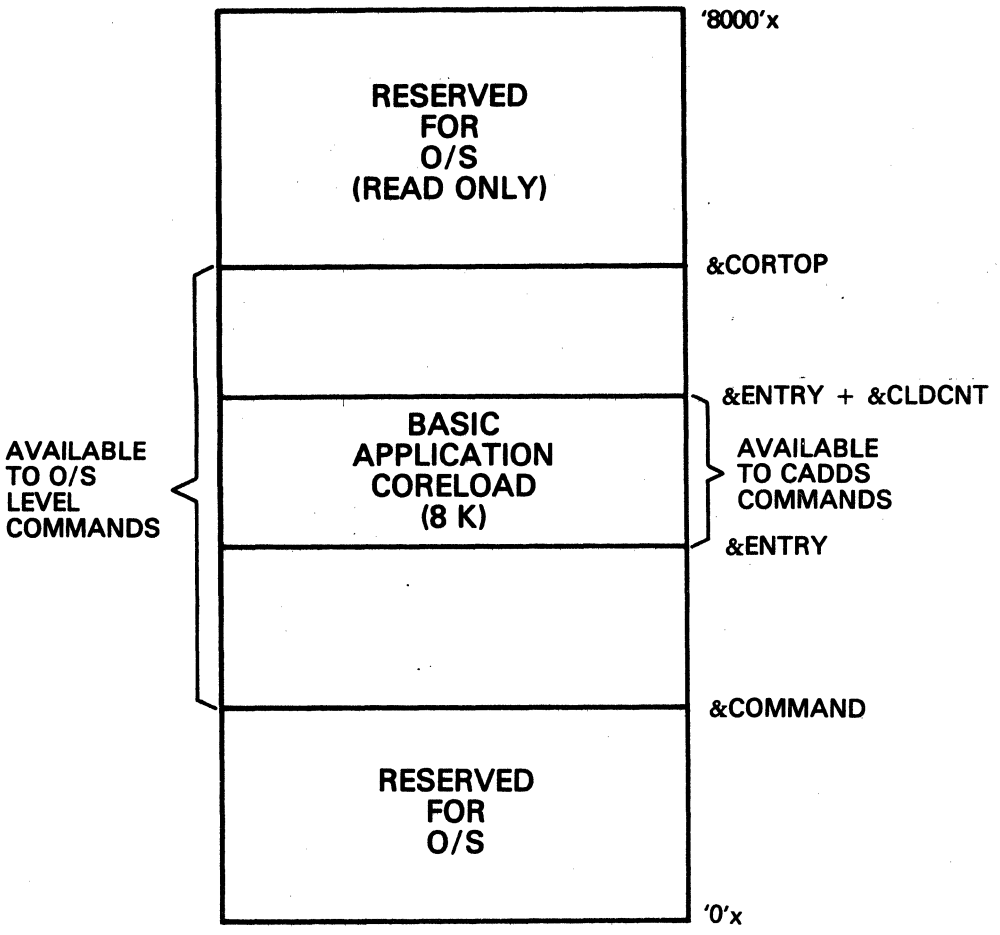
  Each unbundled command is housed in its own disc file. For CADDS commands, these files are named CADDSAUX.Verb.Noun. Each unbundled CADDS command consists of one or more overlays in a CADDSAUX disc file. These overlays are assigned negative coreload numbers. The first overlay in a CADDSAUX file is −1, the second is −2, and so on.

- Referencing Code

  Unbundled coreloads can reference all bundled coreloads. They can also reference other coreloads in their CADDSAUX file. They may not, however, reference other CADDSAUX files, nor can bundled coreloads reference unbundled code. Since unbundled code cannot be referenced from outside its own file, utilities should not be written as unbundled code.

Coremap

When a task is logged-in, it is allocated 32K words of address space, Of this space, 10K is shared and 22K is unique to the task.



Sample Bundled Makefile

```
1!*********************************
2!*
3!****ABC.CLD9000.&BCD.CLD9001
4!*
5!*********************************
6!*
7!LOADSYM INSERT.SYMFILE.STANDARD
```

Replaces the current loader symbol table with symbols defined in INSERT.SYMFILE.STANDARD. Use this insert to reference standard application symbols in every CADDS Make.

```
8!EQU &MYCLD 01
```

Sets the symbol &MYCLD equal to 01 (the last three digits of the coreload number converted to hex).

```
9!*
10!CORORG &ENTRY
```

Defines the lowest address in core where data will be written to. The symbol &ENTRY was given a value in: INSERT.SYMFILE.STANDARD.

```
11!*
12!BLOCK &ENTRY,&ENTRY + &CLDCNT
```

Symbolic definition of address areas

```
13!*
14!LOAD ABC.CLX9000/CLX9001
15!LOAD     (Your routines)
   .
   .
   .
16!LIB LOADLIB.CADDSLIB
17!LIB LOADLIB.FORTLIB
18!LIB LOADLIB.OSLIB
```

These LIB statements are used with CADDS 4 to reference load libraries containing standard routines. With CADDS 3, use the following load libraries instead:

```
16!*************************
17!LIB LOADLIB.FORTLIB          (CADDS 3 Load Libraries)
18!LIB LOADLIB.OSLIB3
19!*************************
20!FILENAME CADDS.OVLY
```

All bundled code is written to the file CADDS.OVLY.

```
21!WRITE &MYCLD*NSCL+&DLB9,&ESICNT,&ESILOC,&CLDCNT,&ENTRY
```

Tells the loader to write to a disc file, giving the location, the offset, and the starting core location

- Executing Bundled Code

To access the Makefile (O/S level), type:

```
n>LOAD CLD9000.CLD9001/DEBUG=ABC ↵
```

DEBUG=ABC tells the loader to look for the ABC version. This should be used in the initial debugging phase to prevent the loader from writing to CADDS.OVLY. When the program is successfully loaded:

```
Hexnum WORDS AVAILABLE IN BLOCK

WRITE val1,val2,...val5 PERFORMED ON FILE
CLD9000.CLD9001

SYMBOL TABLE SAVED IN FILE
CLD9000.&SYM.CLD9001hexnum
```

is output. Next, enter CADDS:

```
n>CADDS ↵
#SEL DEBUG DIR ABC ADD ABC//NLEV ↵
```

Sets up debugging overlays. To test the code:

```
#TEST CLD 9001 ↵
```

Sample Unbundled Make File

```
1!*************************************
2!*
3!****ABC.CLDAUX.TRYONE.&BCD.CLDA01
4!*
5!*************************************
6!*
7!LOADSYM INSERT.SYMFILE.STANDARD
8!EQU &MYCLD 00
```

Sets the symbol &MTCLD equal to 00, meaning this is the first module in the file CADDSAUX.

```
9!*
10!CORORG &ENTRY
11!*
12!BLOCK &ENTRY,&ENTRY + &CLDCNT
13!*
14!LOAD ABC.CLXAUX.TRYONE/CLXA01
```

TRYONE is a verb-noun combination to describe the function of the routine. It should be unique to the catalog CLXAUX.

```
15!LOAD        (Your routines)
    .          (Order not important)
    .
    .
16!*
17!LIB LOADLIB.CADDSLIB
18!LIB LOADLIB.FORTLIB
19!LIB LOADLIB.OSLIB
```

With CADDS 4, these standard load libraries should always be specified in this order. For CADDS 3, use the following load libraries (order is important):

```
16!*************************
17!LIB LOADLIB.FORTLIB        (CADDS 3 Load Libraries)
18!LIB LOADLIB.OSLIB3
19!*************************
20!*
21!FILENAME CADDSAUX.TRYONE
22!WRITE &MYCLD*NSCL,&ESICNT,&ESILOC,&CLDCNT,&ENTRY
```

Tells the loader to write to a disc file, giving the location, the offset, and the starting core location

- Executing Unbundle Code

At system level, type:

```
n>LOAD ABC.CLDAUX.TRYONE.CLDA01 ↵
```

The message:

```
Hexnum WORDS AVAILABLE IN BLOCK

WRITE val1,val2,...val5 PERFORMED ON FILE
ABC.CLDAUX.TRYONE.CLDA01

SYMBOL TABLE SAVE IN FILE
ABC.CLDAUX.TRYONE.&SYM.CLDA01
```

indicates success. To test the routine:

```
n>CADDS ↵
#RUN PROG CADDSAUX.TRYONE ↵
```

Executing Code at O/S Level

For O/S level commands, the basic coreload begins at &COMMAND and extends to &CORTOP. (Refer to the coremap description elsewhere in this section.) O/S level commands should not be loaded with the DEBUG or OVCAT options of the loader.

To execute an O/S level command:

```
n>LOAD Overlayname ↵
```

where Overlayname is the name of the overlay containing the command and its affiliated routines.

The message:

```
Hexnum WORDS AVAILABLE IN BLOCK

WRITE val1,val2,...val5 PERFORMED ON FILE
Overlayname

SYMBOL TABLE SAVED IN FILE
Overlay.&SYM.name
```

indicates success. To execute the routine:

```
n>CADDS ↵
#RUN PROG CADDSAUX.TRYONE ↵
```

COMMAND EXECUTION

Command Table Search

When the user enters a command at system level, command tables are searched in the following order:

1. SYSUSERCMTBXXXX (if activated by the USERCMTB command).
2. SYSCMTB (is added to by the SYSCMTB or EDITCMTB command).
3. The system command table.
4. The SYSCOMMAND catalog is searched for an overlay file.
5. The CVSCOMMAND catalog is searched for an overlay file.
6. If the command string is a filename, the system will attempt to run the DD command with the file as input.
7. If the DD command fails and the file is an overlay created with the loader CWRITE command, the system will attempt to RUN the command.

A command listed in more than one COMMTAB is defined by the first entry found.

Sequence of Execution

Once a command definition has been located, the system begins executing the command. Command execution follows this sequence:

1. The memory command area is set to core constant ('C0F'x).
2. There is a queue for tasks awaiting processing by the CPU. The task is initially assigned the highest allowable priority in this que.
3. If core count is nonzero, core count words are read in from command disc location or overlay file into core location.
4. If the command is in an overlay file, that file will be closed.
5. The system scan routines (INITCHAR, NXTCHAR) are set at the first nonblank character following the command name on the O/S command line.
6. The system starts execution for the task at the starting address.

Cross-Referenced Command Processing

The search for a cross-referenced command begins in the COMMTAB of the cross-referenced entry.

A command may not be defined by more than one level of cross-referencing. Therefore, all cross-referenced entries are ignored in the search for a cross-referenced command.

The starting address for a cross-referenced command is incremented by the relative start location.

Section 3
DEBUGGING

Section 3
DEBUGGING

SUBROUTINE TRACE

The Subroutine Trace tool helps to isolate program bugs by allowing the user to monitor subroutine calls. As the program runs, calling and returning addresses of subroutines are displayed. Normal program execution is unaffected by the Subroutine Trace tool.

Tracing and Debugging

The Subroutine Trace tool is closely related to the Application Debugger. These two features speed identification and correction of program problem areas. When a task is run with the Debugger or Trace tool, one task worth of additional memory space is required. If user address space is full, one task must be logged out to run the Subroutine Trace or Debugger.

Using the Trace Tool

Prefix an O/S level command line with the keyword TRACE. During execution of the object file, a "trace map" is output. This map indicates the calling and returning sequence for subroutines within a program. Each level of indentation on the trace map indicates a subroutine being called from within another subroutine. As the trace map is displayed, normal command output will be interspersed with trace output.

Memory Requirements

A task that runs with the Trace tool or Application Debugger requires one task of additional memory. If the system has insufficient free memory, TRACE and DEBUG won't be enabled unless one task logs out.

Controlling Trace Output

There are two ways to toggle trace output once the subroutine trace has been invoked:

- From the COMDEV, CTRL-T allows the user to start and stop trace output without affecting program execution.
- Within a program, the System Overlay Routine SUBTRACE can be called with a literal argument of ON or OFF to toggle trace output on or off. A call to SUBTRACE.('OF') at the beginning of the command will suppress trace output. If tracing is not enabled, a call to SUBTRACE has no effect. To find trace status, use CHKTRACE. The function return will be either 'ON' or 'OF', to indicate whether the trace is on or off.

SYMBOLIC REFERENCES

Unless the NOTRACE option is used, the loader will automatically save the symbol file for access by the Subroutine Trace tool and the Application Debugger. This enables the trace map to include symbolic entry point names alongside subroutine calling addresses.

Symbol Filenames

Symbol files from the TRACE or DEBUG options of the loader are stored by the file manager under the following naming conventions:

- Symbol filenames are derived from the name associated with the loader keyword FILENAME.
- &SYM is inserted as the second to last level of the symbol filename.
- Either VAL1 from the loader WRITE and OVWRITE commands, or 0000 from the loader CWRITE command is appended to the last level of the complete filename.

EXAMPLES

- Loader contains:

```
FILENAME RENUM
WRITE 00E9,0,0,2000,6000
```

  Symbol filename:

```
&SYM.RENUM00E9
```

- Loader contains:

```
FILENAME SYSCOMMAND.COMPARE
CWRITE 4000,2000,5000
```

  Symbol filename:

```
SYSCOMMAND.&SYM.COMPARE0000
```

Note

See LOADER documentation for additional information.

Overlays

Programs that are divided into memory overlays must be modified for tracing. This requires two new calls:

- Instead of FREAD, use OREAD.
- The Make file must use LIB LOADLIB.OSLIB.

Neither of these changes will effect performance.

Sample Output without Symbolic References

```
Commands                                  Comments

LOAD MAKE.COMMAND.COMPARE/NOTRACE         ! Load for command

TRACE COMPARE TEST1,TEST2                 ! Prefix TRACE to
                                          ! command line

CALLING TO 2000                           ! Main routine
    CALLING TO 21F3                       ! Call to a routine
                                          ! loaded at 21F3
     RETURN TO 200D                       ! Return to main routine
     CALLING TO 21F3
     RETURN TO 2016
    CALLING TO 27D2                       ! Calls routine loaded
                                            at 27D2
        CALLING TO 2A42                   ! Calls routine loaded
                                            at 2A42
        RETURN TO 27E5                    ! Return to location
                                            27E5
    RETURN TO 2024                        ! Return to main routine
    CALLING TO 27D2
        CALLING TO 2A42
        RETURN TO 27E5
    RETURN TO 2032
```

Sample Output with Symbolic References

```
Commands                                  Comments

LOAD MAKE.COMMAND.COMPARE                 ! Load for command

TRACE COMPARE TEST1,TEST2                 ! Prefix TRACE to
                                          ! command line

CALLING TO 2000 COMPARE                   ! Main routine
    CALLING TO 21F3                       ! Internal subroutine
    RETURN TO 200D
    CALLING TO 21F3                       ! Internal subroutine
    RETURN TO 2016
```

Sample Output with Symbolic References (Continued)

Commands	*Comments*
CALLING TO 27D2 GETFIL	
CALLING TO 2A42 FMEXPNM	
RETURN TO 27E5	! Return from FMEXPNM to GETFIL
RETURN TO 2024	! Return from GETFIL
CALLING TO 27D2 GETFIL	! to COMPARE
CALLING TO 2A42 FMEXPNM	
RETURN TO 27E5	
RETURN TO 2032	

APPLICATION DEBUGGER

A debugger helps to isolate program bugs by allowing the user to monitor and change program execution. By setting breakpoints, the programmer can divide a program into segments and process one segment at a time. When a breakpoint is reached, all processing halts. Values of program variables may be examined and reset at any breakpoint.

The Application Debugger is closely related to another system feature, the Subroutine Trace tool. These two features speed identification and correction of program problem areas.

To Access the Debugger

The debugger is invoked by prefixing any system command with the keyword DEBUG, this gets the user to debugger command level. On initial entrance, the debugger displays special notes as to command changes or new features. The prompt "DBG]" indicates that the debugger is at command level, ready to accept commands.

Features

The features included apply variously to syntax, toggles, following a break, and to local variables.

- Syntax

  Terminate commands with a carriage return or a semicolon. The semicolon allows several commands to be strung together on one line. Most commands accept multiple arguments separated by commas and then process each argument or group of arguments individually. In this example:

  BS GETLINE,PUTLINE,GETCLOSE

  sets a breakpoint at the address of each subroutine name.

- Toggles

  CTRL T is used to enable and disable Subroutine Trace output. When a program is executing and the user would like to return to the debugger prompt, a CTRL D will force a break to debugger command level at the next subroutine entry or exit.

- Following a Break

  After issuing debugger commands, the user may resume program execution with one of three commands: "BX" forces a break at the next subroutine entry or exit. P means proceed, or PN for proceed, not checking. P checks for subroutine breakpoints during program execution and returns to command level when one is found. PN disables the debugger and continues normal execution; breakpoints are ignored. CTRL D will re-enable the debugger and return to debugger command level at any time.

- Local Variables

  Local variables can only be accessed from the subroutine containing the current breakpoint. To examine a local variable, compile the subroutine using the TRACE option. If the routine was loaded with the NO TRACE option, reload. Then activate the full filename using the AF command (See Debugger Commands). Prefix local variable symbol names with a period.

Restrictions

Restrictions apply as indicated below.

- Hexadecimal Numbers

  Use hex numbers to represent addresses and values. A hex number whose first digit is A-F must be preceded by a zero. For example, 'FFFF'x should be entered as '0FFFF'x.

- Breakpoints

  Only breakpoints set at a subroutine entry or exit point will cause an actual break in program execution.

- Address Space

  Breakpoints may not be set outside of application address space. System Resident Routines and System Overlay Routines are outside of application address space. Breaks may not be set within these routines.

- No-Wait I/O

The Debugger automatically resets no-wait I/O to wait I/O.

DEBUGGER COMMANDS

These commands apply to general control, toggles, and to breakpoints.

General Control

P	Proceed: Continues program execution, checking for breakpoints.
PN	Proceed, not checking: Continues program execution without checking for breakpoints.
BX	Forces a break at next subroutine entry or exit point, regardless of whether a breakpoint was previously set.
QUIT	Returns the user to a system level prompt.
H	Displays debugger commands.
?	Same as H.
;	Used to separate commands.

Toggles

Both CTRL D and CTRL T act as toggles.

CTRL D	Forces a break to debugger command level at the next subroutine entry or exit point.
CTRL T	ON/OFF switch for subroutine tracing.

Breakpoints

Breaks will only be executed at subroutine entry points.

LB	Lists breakpoints.
BAaddress1,address2...	Sets breakpoints at address1,address2,...
BS name1,name2...	Sets breakpoints at entry points to subroutines name1,name2,...
CA address1,address2..	Clears addresses from breakpoint list.
CS name1,name2...	Clears subroutine entries from breakpoint list.
CL	Clears all breakpoints from list.

Controlling the Execution of Breakpoints

With the "count" and "reset" options, you can set the debugger to execute breaks at specified intervals. COUNT and RESET must be decimal integers.

BS name1[(count[,reset])],name2[(count[,reset])]...
BA address1[(count[,reset])],address2[(count[,reset])]...

count An initial value that gets decremented each time the subroutine is called; at zero, a break will be be triggered. A value of zero or one means "break at every call".

reset Count is reset to this value after the first break to the debugger. The default for reset is zero.

Several examples follow:

BS FOO(5)	Breaks at the fifth call to subroutine FOO and then every call after the fifth.
BA 2345(3,3)	Breaks on every third call to the subroutine at address 2345.
BS DBNEG(1,5)	Breaks on the first call to DBNEG and every fifth call after the first.

Dump Commands

Three examples are described.

D address,length	Dumps memory, starting at address, for the specified length.
D address1 > address2	Dumps memory from address1 address2.
DV name	Dumps the contents of memory (addressed by a symbolic name).

Setting Memory

Setting memory through the use of SA and SV.

SA address = hex value 1, value 2...valuen	Sets memory address to hex values.
SV name = hex value 1, value 2...valuen	Sets variable with specified name to hex values.

Searches

Searches for a hex value within the address bounds:

SR value IN address,length.
SR value IN address1 address2.

Miscellaneous

AV name	Returns address of variable.
AC command1;command2	Initiates automatic command mode: This string of commands will be executed at each breakpoint. AC with no arguments resets the command. If looping occurs, CTRL D will force a return to the debugger prompt.

Local Variables

Local variable names must be prefixed with a period.

AF filename	Filename is the routine's full filename. Use the trace option for compiling the routine.
PF	Print activated filename.

Section 4
BRIEF SUBROUTINE DESCRIPTIONS

Section 4
BRIEF SUBROUTINE DESCRIPTIONS

This section contains a brief description of subroutines and subroutine utility packages available to the application programmer. Each subroutine is described with a brief summary of its purpose. Subroutines are primarily grouped as to function, and secondarily, sorted alphabetically. More complete descriptions are listed alphabetically in Section 5.

LINKAGE FOR ROUTINES

A reference to a system routine can be resolved by referencing one of three places in the operating system. To ensure complete linkage for your code, reference each of these places every time you load a program. Instead of explicitly loading a system routine, use the inserts and libraries mentioned below to resolve any references to the routine.

Note

User-written routines should be explicitly loaded in the Make.

System Resident Routines

System Resident routines are permanently resident in core and available to all tasks. They are frequently used routines that often require access to system utilities. Several tasks may access a System Resident routine at the same time. System Resident routines are called explicitly or implicitly from within user code (for loader inserts, see Section 2).

System Overlay Routines

These routines are referenced in the System Overlays (SYSOVLY). Overlay Routines are automatically read in from disc when not in memory.

These routines are defined in the standard system inserts (see Section 2).

System Library Routines

These routines do not have to be explicitly loaded by the programmer. They do take up space in the user task area, however. Before terminating a block in the Make, specify:

```
LIB LOADLIB.FORTLIB
LIB LOADLIB.OSLIB
```

or, for CADDS 3,

```
LIB LOADLIB.OSLIB3
```

These commands, included in this order, should resolve any unresolved references in the program.

Brief Subroutine Descriptions

ROUTINES LISTED BY FUNCTION

Changes to these routines will be reflected in the on-line documentation. To access on-line documentation from O/S level type:

HELP SUBROUTINE

or

HELP SYSROUTINE

File Manipulation

The following are low-level file manager routines.

COPEN	Opens a file for create or create with supersede.
MOPEN	Opens a file for modification.
ROPEN	Opens a file for reading.
FREAD	Reads an open file.
OREAD	Reads an overlay into memory (Used with Subroutine Trace and Application Debugger, See Section 3).
FWRITE	Writes to an open file.
CLOSE	Closes a file.
DELETE	Low-level FM routine to delete a file.
CHKPROT	Validates access rights to a file or command.
CHKUPROT	Verifies caller's default protection group for access to a file or command.
RENAME	General file manager utility routine.

File Utility Packages

GETFILE	GETFILE	Opens a text file for reading.
	GETLINE	Reads the next line from a text file.
	GETLINEB	Equivalent to GETLINE with a byte offset in the output.
	GETCLEAR	Marks the file status block of an open text file to indicate that the buffer is no longer available. The next GETLINE call will do an FREAD.
	GETMARK	Saves a position in a text file.
	GETPOS	Restores a saved position within a text file.
	GETSTART	Moves pointer to the beginning of a text file.
	GETCLOSE	Closes a text file opened with GETFILE.

PUTFILE — Writing a text file.

PUTFILE	Opens a text file for output.
PUTABORT	Aborts text file being written by PUTFILE.
PUTLINE	Writes a line to a text file.
PUTLINEB	Equivalent to PUTLINE with a byte offset.
PUTCLEAR	Marks the file status block of an open file to indicate that the text file buffer is no longer available.
PUTCLOSE	Closes a text file opened with PUTFILE.

READFILE — Used for reading words from a binary sequential file.

READFILE	Opens a sequential text file for reading. The standard sequence is to open with READFILE, read blocks of words with READBLOK, and close with READCLOS.
READBLOK	Reads the next block of words from a binary sequential file.
READSECT	Reads a block of words from a file opened for sequential word access.
READMARK	Returns the current position in a binary sequential file.
READPOS	Sets a new position in a binary sequential file.
READTOP	Moves pointer to beginning of a binary sequential file.
READCLOS	Closes a file opened with READFILE.

WRITFILE — Used primarily for writing words to a binary sequential file.

MODIFILE	Opens a binary sequential file for modification.
WRITFILE	Opens a binary sequential file for write. The standard sequence would be to create with WRITFILE, add blocks of words with WRITBLOK, and close with WRITCLOS.
WRITBLOK	Writes a block of words into a binary sequential file.
WRITCLOS	Closes binary sequential file opened with MODIFILE or WRITFILE.

READBFIL — Reads bytes or characters from a binary sequential file.

READBFIL	Opens a binary sequential file for byte-oriented input.
READBYTE	Retrieves the next block of bytes from a binary sequential file.
READBCLS	Closes the input channel of a binary sequential file opened with READBFIL.

WRITBFIL — Writes bytes or characters to a binary sequential file.

WRITBFIL	Opens a binary file, for writing with byte-oriented counts.
WRITBYTE	Writes a record to a file opened by the routine WRITBFIL.
WRITBCLS	Finishes writing and closes a binary file opened with WRITBFIL.

Processing a Catalog — There are two subroutine packages available for executing an FM catalog structure tree walk: CATWALK and TRAVERSE.

CATWALK — Calls a user-defined subroutine to process nodes of a catalog. There are optional arguments for screening out various types of files, but processing of individual files remains under the control of CATWALK.

CATBREAK	Recovers, if CATWALK is interrupted.
CATWALK	Processes selected nodes of a catalog with a user-defined subroutine.

TRAVERSE — The TRAVERSE package processes a catalog by returning information about nodes and files. The actual processing of files is left to the discretion of the calling routine.

TRAVERSE	Initializes the TRAVERSE package by setting up the catalog to be traversed and the information to be returned for each file.
NEXTNODE	Returns the next node or file to be processed by the calling routine.
ABORTRAV	Terminates the tree walk and "cleans up" after error returns or after the entire catalog has been traversed.

Deleting a File

DELETEXT	Deletes a text file.
DELFIL	Deletes a file and any empty catalogs above it.

Filename Manipulation

FMCNTRNM	Removes the next to last level of a filename.
FMEXPNM	Expands a filename by adding a level before the last level.

Memory Page Manipulation — System memory consists of 2K pages. Each page has a system memory page identifier (SPID). Users refer to a memory page by specifying its SPID. From the viewpoint of the application programmer, the SPID is a 16-bit integer.

ALLOCATION

The two allocation routines that follow permit the user to request memory pages for a task, or to return pages from the task to the system. The system limits the number of pages that a single task can own. Once they are allocated, however, the user becomes the owner of the pages.

ALLOPG	Allocates memory pages to user.
FREEPG	Returns memory pages to system.

USER MEMORY

Each user has an address space of 32K words; this spaces is divided into sixteen 2K pages. The system manages a date structure called a "page frame" that describes which sixteen 2K pages are mapped to the 32K user address space. The two routines that follow permit the user to modify his page frame by either placing a page into, or removing a page from, any 2K page slot in his page frame.

GETPG	Returns system page ID and permission associated with a slot in the user's page frame.
SETPG	Modifies a page slot of the user's page frame.

PROTECTION

The user controls logical access to his pages. Protection access can be controlled with the following routines:

GETPGP	Reads the protection attributes of a page.
SETPGP	Modifies the parameters of a system page.

Date and Time

CMPDAT	Compares two dates (in system date and time format).
FMIDAT	Scans date (and time) from input stream, and converts them to system date and time format.
FMTDAT	Converts a date from system date and time format (as found in file entries) to printable format.
GETDAT	Returns the current date and time (system date and time format).
TIME	Returns the current date and time in a four-word array.

Task-Related Functions

GETTASK	Gets the number of the current task.
GETTASKF	Calls GETTASK from a FORTRAN program
HIBERN8	Deactivates task for specified time.
CLEARCOM	Fills the command buffer with blanks.
SETSTRG	Puts an eight-character string into the task message area.

Errors

ERROR	System error routine that returns control to the O/S.
PERROR	Prints an error message from a message file when given an error number, then returns to the calling routine.
RETERR	Returns the last system error number.

Peripheral Devices

Includes task functions for attaching the following: card reader, paper tape devices, and magnetic tape drive.

- Task Functions

ATTACH	Attaches a device to a task.
UNATTACH	Detaches a device from a task.

- Card Reader

CARDIN	Reads a card from CARDEV.

- Paper Tape Devices

CKPPTD	Checks whether a punch paper tape unit (PPTDEV) is assigned to a task.
CKRPTD	Checks whether a read paper tape unit (RPTDEV) is assigned to a task.
GETPPTD	Attaches PPTDEV to a task.
GETPUNCH	Returns parity option of PPTDEV.
GETRPTD	Attaches RPTDEV to a task.
PNCHLDR	Punches a leader on PPTDEV.
PPT	Outputs words to PPTDEV.
PPT1	Outputs one character to PPTDEV.
PPTN	Outputs a string of bytes to PPTDEV.
RPT	Inputs words from RPTDEV.
RPT1	Inputs one character from RPTDEV.
RPTN	Inputs bytes from RPTDEV.
SETPUNCH	Sets the parity option for task PPTDEV.

- Magnetic Tape Drive

TAPE	Processes a tape request according to keyword (KYWD).
TAPENW	Processes a tape request according to ARGLST with the no-wait entry.
TESTTAPE	Validates a tape unit task name.
TREAD	Reads a physical record from magnetic tape (unpacks if necessary).
TREADNW	No-wait version of TREAD.
TWRITE	Writes words on magnetic tape (unpacks if necessary).
TWRITENW	No-wait version of TWRITE; writes words on magnetic tape and returns I/O flag.

Subroutine Trace

These routines are used with the subroutine trace (see Section 3).

SUBTRACE	Toggles trace map output.
CHKTRACE	Gets status of trace map output.

Scanning

INITCHAR and NXTCHAR are automatically called to set up scanning of the command line. The two routines must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).

Note

All scan routines call ERROR when an error is detected.

INITCHAR	Sets up an array and character count for all system scan routines.
NXTCHAR	Gets the next character to be scanned, along with its type.

VALUES AND CHARACTERS

DBHXNM	Converts ASCII string to double precision integer.
DBHXLST	Scans a hex bounds list from the input stream.
DBINT	Scans a double precision decimal integer from the input stream.
DBINTLST	Scans a double precision integer list from the input stream.

	HEXLST	Scans hex bounds list from the input stream.
	HEXNUM	Scans a hex number from the input stream.
	IDENT	Scans for an eight-BCD-character identifier from the input stream.
	INT	Scans a decimal integer from the input stream.
	INTLST	Scans a decimal bound list from the input stream.
FILE AND CATALOG NAMES	INITNM	Initializes scanning of an FM NAME LIST.
	NXTNAM	Gets next name from FM NAME LIST.
	FMIDNF	Scans for a name (up to 20 characters) from the input stream (use with FORTRAN routines).
	FMIDNT	Scans for a name (up to 20 characters) from the input stream (use with TPL routines).
	FMNAME	Scans for a catalog name from the input stream.
DATE	FMIDAT	Scans date (and time) from input stream, and converts them to system date and time format.
Input/Output	This section includes standard and no-wait I/O.	
	TOGGLE	Manipulates COMDEV and HARDEV toggles.
INPUT	STYPIN	Functions like TYPIN, but uses options specified by the program.
	TYPIN	Inputs a sequence of characters terminated by a carriage return from either the COMDEV or an execute file.
	STYPEOK	Tests task COMDEV for "OK" response to specified message.
	TYPEOK	Tests for an "OK" response to a specified message. Response may come from task COMDEV or execute file.
OUTPUT	GLP	Outputs line of characters to device.
	LP	Outputs line of characters to COMDEV and the HARDEV.
	TYPE	Outputs a line of characters, inserting a carriage return and a line feed at the end of the line.
	TYPEDBI	Outputs double precision integer in decimal format.
	TYPEINT	Outputs an integer in decimal format.

	TYPOUT	Outputs a line of characters without inserting carriage returns or line feeds.
	TYPEDBHX	Outputs double precision integer in hexadecimal format.
	TYPEHEX	Outputs an integer in hexadecimal format.
NO-WAIT I/O	SRTESTIO	Tests an IOFLAG for I/O completion and returns error numbers.
	TESTIO	Tests an IOFLAG for I/O completion and calls system error handler when I/O errors are detected.
	SRWAITIO	Changes a NOWAIT I/O request to a WAIT I/O request and returns error numbers.
	WAITIO	Changes a NOWAIT I/O request to a WAIT I/O request.
Breaks and Labels	CHKBREAK	Checks whether CHKINTR, CHKQUIT, or CHKSTOP have detected a break.
	CHKINTR	Checks for interrupt and waits or sets appropriate one-time switch in the task block. Handles all (ESC) characters.
	CHKQUIT	Checks for interrupt (ESC) action based on next character input. Handles only Q, N, and K.
	CHKSTOP	Checks for interrupt (ESC). If found, performs a quit.
	CLRBREAK	Clears a task break flag.
	LBLGO	Transfers control to task system label or statement label (see LBLSET).
	LBLSET	Sets up a statement label or task system label for LBLGO transfer.
	SETBREAK	Enables flag-setting for CHKBREAK and disables task inter-quit process.
	GETLABEL	Accesses and saves a task system label.
	PUTLABEL	Restores a task system label saved by GETLABEL.
	RESETLBL	Restores a task system label to its original state.

Double Precision Integers

This section discusses subroutines for manipulating and comparing double precision integers.

- Addition and Subtraction

ADJUST	Increments a double word integer by a single word integer.
DBADD	Adds two double word integers.
DBDCR	Decrements a double precision integer.
DBINC	Increments a double precision integer.
DBSUB	Subtracts two double word integers.

- Division And Multiplication

DBDIV	Divides double precision integers.
DIVUS	Divides unsigned integers.
DBMUL	Multiplies unsigned single word integers, giving a double word integer.
DBNEG	Reverses sign of double word integer.
LABS	Finds absolute value of a double integer.

- Manipulation and Comparison

DBMAX	Finds the larger of two double integers.
DBMIN	Finds the smaller of two double integers.
DBLSH	Executes a double word left shift.
DBRSH	Executes a double word right shift.

Comparison

CMBYTF	Compares bytes in two arrays (FORTRAN version).
CMBYTT	Compares bytes in two arrays (TPL version).
COMPN	Compares words in two arrays.
COMPNAM	Compares two eight-BCD-character names.
COMPUS	Compares two unsigned integers.
DBCMPR	Compares two unsigned double integers.
MAXIMUM	Finds larger of two unsigned integers.
MINIMUM	Finds smaller of two unsigned integers.
TSTZERO	Tests if a double precision integer equals zero.

Conversion

DBHEXBCD	Converts double precision integer to text string using the hexadecimal radix.
DBINTBCD	Converts a signed double precision integer to BCD.
DBLE	Converts real to double precision real.
DFLOAT	Converts an integer to a double precision real.
DFLOATL	Converts a double integer to a double precision real.
FLBCD	Converts a floating point number to an ASCII text string.
FLOATL	Converts a double integer to a real.
HEXBCD	Converts a hex word to BCD.
IFIXD	Converts a double-precision real to a single-word integer.
INTBCD	Converts an integer to a character string.
ISNGL	Converts a double integer to an integer.
LFIX	Converts a real to a double integer.
LFIXD	Converts a double precision real to a double integer.
OCTBCD	Converts an integer to an octal text string.
SNGL	Converts double-precision real to real.

Array Manipulation

The functions of filling an array, byte and character manipulation, and the moving of data between arrays are included.

- Filling An Array

FILL	Fills a word array with a given value.
FILLBYTT	Fills a byte array with a given value.
FILLCHRT	Fills a character array.
FILLDB	Fills a double precision integer array with a given value.
FILLF	Fills a real array with a given value.

- Bytes and Characters

STBYTF	Stores a byte (FORTRAN version).
STBYTT	Stores a byte.
STCHRF	Stores a character (FORTRAN version).
STCHRT	Stores a character.
LDBYTF	Loads bytes (FORTRAN version).

LDBYTT	Loads bytes (TPL version).
LDCHRF	Loads characters (FORTRAN version)
LDCHRT	Loads characters (TPL version).

- Moving Data Between Arrays

MOV	Moves integer data from one array to another.
MOVB	Moves integer data from one array to another, starting from the end of each array.
MOVD	Moves double precision real data from one array to another.
MOVEWORD	Moves a character string from one array to another.
MOVF	Moves real numbers from one array to another.
MOVL	Moves double precision integers from one array to another.
MVBYTF	Moves bytes (or characters) from one array to another (FORTRAN callable).
MVBYTT	Moves bytes (or characters) from one array to another (TPL callable).

Stack Manipulation

IPDL	Initializes a push-down list.
POP	Pops a word off a push-down list.
POPN	Pops a block of words off a push-down list.
PUSH	Pushes a word onto a push-down list.
PUSHN	Pushes a block of words onto a push-down list.

Bit Manipulation

GETBIT	Returns the value of a given bit.
TSTBIT	Returns the value of a given bit.
PUTBIT	Turns a bit ON or OFF.
SETBIT	Turns a bit ON.
CLRBIT	Turns a bit OFF.
GETFLD	Retrieves the value in a field and returns it as a right-justified integer.
PUTFLD	Puts a value into one field of a bit string.

Sorting Records

Sort routines are divided into two categories; non-stable sorts and stable sorts.

- Non-stable sorts (heap,shell) are generally faster, but they may change the original order of records with identical sort keys.
- Stable sorts (straight insertion, selection) are generally slower, but they maintain the order of records with identical sort keys.

HEAPSORT	Sorts an internal file of single word records using a heap sort.
INSESORT	Sorts an internal file of single word records using a straight insertion sort.
INSESRTN	Sorts records using a straight insertion sort.
SELESORT	Sorts an internal file of single-word records using a selection sort.
SHELSRTN	Sorts records using a shell sort.

Miscellaneous

GETSTAT	Reads task or system statistics blocks.
XEQTCOMM	Executes a system command passed as a subroutine argument.
CHKSUM	Computes CHECKSUM for an array.
HEXDMP	Debugging aid that dumps hex words from an array.

Section 5
SUBROUTINES

Section 5
SUBROUTINES

This section describes the conventions used and an alphabetic presentation of subroutines.

ROUTINE DESCRIPTION CONVENTIONS

There are several headings for each subroutine description. The most common types are: SYNTAX, INPUT, OUTPUT, and FUNCTION RETURN. These headings have the following meanings:

Syntax

In relation to the calling code, the subroutines mentioned in this manual can be treated as either functions or external routines.

To call an external routine:

FORTRAN

```
CALL Subroutinename (Arg1,Arg2,..Argn)
```

TPL

```
E:E Subroutinename.(Arg1,Arg2,..Argn)
```

To call a function:

FORTRAN

```
Variable = Functionname(Arg1,Arg2,..Argn)
```

TPL

```
Variable = Functionname.(Arg1,Arg2,..Argn)
```

TPL can call a function with no arguments but FORTRAN cannot. Arguments (Arg1,Arg2,..Argn) are mandatory to the function or subroutine call. The given call sequence is also mandatory, since the designation of arguments as either INPUT, OUTPUT, or both, cannot change.

Input

INPUT refers to arguments that accept values for processing by the routine. All routines treat input arguments as constants and output arguments as variables. If no input arguments are mentioned, the subroutine does not accept input.

Output

OUTPUT can assume one of two forms:

- Subroutine arguments can receive values as output. These arguments are processed by the subroutine
- Some subroutines output data to the COMDEV; this output may also be routed to a HARDEV or another device by setting the appropriate toggle.

OUTPUT should not be confused with FUNCTION RETURN. If no output arguments are specified, then the only way to pass values from the called subroutine to the calling routine is via the function return. Some arguments may be both INPUT and OUTPUT.

Function Return

If the routine is called as a function, then this value will be passed to the variable on the left side of the assignment statement within the calling code. In FORTRAN, it is important that Variable be of the same type (i.e., integer) as the function.

For Example:

```
TASKNUM = GETTASKF(INUM)
```

UNIVERSAL

Apply to all the routines in this manual.

Bit Numbering

In system subroutines, bits are numbered from left to right, beginning at zero (See Section 6.)

Note

In CADDS application routines, bits are numbered from right to left.

Array Indexing

In this manual, array indices follow FORTRAN conventions. Indexing begins at one, rather than zero as in TPL. To convert indices to TPL, subtract one.

PARTICULAR Apply to some, but not all, routines.

FM Name Format File or catalog names are generally passed to subroutines in in FM Name format, as follows (Word positions are in parentheses):

Word	*Contents*
(1)	Count of bytes in name (including delimiters and terminating character).
(2-n)	Characters in name, terminated by an exclamation point (!), with a period (.) as delimiter between components.

Routines with file manager entry points (ROPEN, COPEN, RENAME, etc.) DO NOT accept a byte count in Word 1 of the filename.

Note

No component of the name can exceed 20 characters, and the entire name, including the (!), must not exceed 80 characters.

Options OPTIONS refers to an array of one or more words. Some words may contain bit set flags; other full words or blocks of words may contain data. The exact format depends on the subroutine. In the syntax of this manual, words are enclosed in parentheses and bits are listed under the "bit" column heading (for more information, see SYSTEM FORMATS, Section 6).

File Status Block The File Status Block (FSB) is an array that displays the status of a file opened with one of the file utility packages (GETFILE, PUTFILE, READFILE, etc.). Whenever the file is accessed with a subroutine from the package the FSB is modified to appropriately reflect changes to the file.

The GETFILE and PUTFILE utility packages, have an FSB of 8 words. For the READBFIL, WRITBFIL, READFILE, and WRITFILE utilities, FSB is 12 words. In every package FSB(1) must be set to one when the file is opened.

Note

To avoid errors and ensure sequential file operations, preserve the integrity of the FSB.

FILE MANAGEMENT

This information applies to programming under the file management facility on CGOS 200.

- The following routines invoke the file management facility:

CHKPROT	CHKUPROT
CLOSE	COPEN
DELETE	FREAD
FWRITE	MOPEN
RENAME	ROPEN

Link these routines using JANMAKE.SYM.FMSYS, or any SYMFILE that includes the references in JANMAKE.SYM.FMSYS (See Section 2).

Note

Avoid these low-level file management routines whenever possible. Instead, use the high-level file manipulation packages (see Section 4).

- File management system calls follow standard FM file naming conventions except for one important difference. Word 1 of the filename should be the beginning of the characters in the name, not the byte count of the name.

- A full file name cannot exceed 80 characters. The character count includes catalog levels, delimiters and the terminating exclamation point.

- All file management errors are in the hexadecimal range C000 to CFFF. (See SYSNEWS.ERROR.FM.)

SUBROUTINE DESCRIPTIONS

The following subroutines are listed in alphabetical order and described in some detail.

ADJUST

SYNTAX — ADJUST(DARGI,SARGI,DARGO)

PURPOSE — Increments a double word integer by a single integer.

INPUT

DARGI	Double word integer addend.
SARGI	Single word integer addend.

OUTPUT

DARGO	Double word integer sum.

ALLOPG

SYNTAX — ALLOPG(SPID)

PURPOSE — Allocates memory pages.

OUTPUT

SPID	System Page ID of the page allocated to the calling task.

COMMENT — When a page is allocated, its logical protection attributes are initialized by calling SETPGP.(SPID,P&RW).

FUNCTION RETURN

0	No errors.
E040	No page available — all system pages are allocated.
E100	No page available — no more pages can be allocated to the calling task.

ATTACH

SYNTAX ATTACH(NAME,TYPE).

PURPOSE Attaches a unit to a task.

INPUT

NAME Two character (one word) name assigned to the unit.

TYPE One of the following:

- Four character (2 word) generic unit name.
- −1 preceding a specific unit number.
- −2 (attaches the default HARDEV).
- −3 (attaches the first COMDEV)

FUNCTION RETURN

0 Either NAME is already in use, NAME is 'CM', or NAME is a bad type.

−1 No units of TYPE available.

1 Attach successful.

CARDIN

SYNTAX — CARDIN(MAX,ARR)

PURPOSE — Reads a card from the CARDEV.

INPUT

MAX	Maximum number of characters to be moved into ARR.

OUTPUT

ARR	Packed array of characters read from the card.

FUNCTION RETURN

−1	Card reader not ready.
0	Read was successful.
>0	Pick error or trouble.

COMMENTS

- Uses a '50'x word global named CARDBUF.
- Unit characteristic words for the CARDEV determine whether card punches are translated into 029 or 026 character code. If the first unit characteristic word is zero (default condition), 029 characters will be returned. If the first word is non-zero, 026 characters will be returned.
- Section Two of the *CGOS 200 Operator Guide* contains a complete list of both card punch character sets.

ERRORS

'FOOE'x	No CARDEV assigned.

CATBREAK

SYNTAX CATBREAK(PDL)

PURPOSE Resets CATWALK globals and closes open files when the execution of CATWALK has been interrupted.

INPUT PDL Push-down stack used in the call to CATWALK.

NOTE CATBREAK should never be called unless the execution of CATWALK has been interrupted.

CATWALK

SYNTAX

CATWALK(CATNAM,OPTIONS,SUBR,PDL,PDLSIZ,DATES)

Calls a user-defined subroutine with the following syntax:

SUBR(IND,NAME,ARRAY)

PURPOSE

Uses a user-defined subroutine (SUBR) to process selected nodes of a catalog. File-screening is optional. If IND = 2, the node has met file-screening criteria.

INPUT

CATNAM Catalog name, preceded by the character count and terminated by an exclamation mark (!).

OPTIONS(1)

Bit Number	*Status*	*Function*
0-9	Zeroes	
10	ON	Specifies method for selecting types of files from CATNAM to be processed by SUBR, as specified in OPTIONS(3).
	OFF	
11	ON	Screen out files, except those before dates(3-4).
12	ON	Screen out files, except those since dates(1-2).
13	ON	Screen out temporary files on IND=2 calls.
14	ON	Tree walk executed left to right, top down.
	OFF	Tree walk executed left to right, bottom up.
15	ON	Walk through all levels.
	OFF	Walk through one level only.

INPUT (Continued)

OPTIONS(2)

OPTIONS(1), BIT 10 OFF:

= 0 Includes all types.

NE 0 Exclude files which do not meet the file-screening criteria, (i.e., IND≠2); type N is specified by setting bit M in this word where M = 15 – N; type 1 means catalogs.

OPTIONS(1), BIT 10 ON:

= N Where N is the number of words (N ≠ 16) in the following bit array. The bit should equal the type of file to be processed by SUBR i.e., bit:

0 ON	FILE TYPE 0
1 ON	FILE TYPE 1
.	.
.	.
.	.
255 ON	FILE TYPE 255.

This bit array must follow immediately after the options input array.

SUBR Subroutine to be called for each node.

PDL Push-down stack to be used by CATWALK of size equal to maximum number of levels * 12.

PDLSIZ Size of push-down stack.

DATES System date and time format:

(1-2) Since date and time — options bit 12 must be ON.

(3-4) Before date and time — options bit 11 must be ON.

OUTPUT

Calls SUBR in this format:

CALL SUBR(IND,NAME,ARRAY)

All arguments are given by CATWALK. SUBR is a user-defined routine that CATWALK invokes for each node in CATNAM. If IND = 2, the file has met screening criteria.

It is the responsibility of SUBR to examine the value of IND and determine how the node will be processed.

IND

0: Start of CATWALK

NAME: Dummy.
ARRAY: Dummy.

1: Node is "left parent" (there are "child-nodes" beneath it in the catalog structure).

NAME: Node name in the same form as CATNAM; zero implies SYSCATLG.
Extended array from ROPEN for this node.

2: Process this node

NAME: Words
(1-42) Node name.
(43-45) Address in buffer for file ID.
ARRAY: Catalog entry for node.

3: Node is right parent

NAME: Node name.
ARRAY: Dummy.

−1: End of CATWALK

NAME: Dummy.
ARRAY: Dummy.

OUTPUT (Continued)

−2: Node is search catalog. Search not performed due to protection or modification. (Replaces 1 and 3 calls for node, and all calls in between.)

NAME: Node name.
ARRAY: Catalog entry for node.

FUNCTION RETURN

0	Normal completion.
'C0XX'x	File management error, where XX represents two hex digits.

COMMENTS

A system overlay containing CATWALK may be loaded by user defined overlays. To use the CATWALK overlay:

- The object code 'LIB.CATWALK.LOADCATWALK' must exist in the task file area.
- The user overlay must begin at &FCTWKCM instead of &COMMAND (both are defined in the loader symbol file JANMAKE.SYM.BASIC).
- The file 'SYM.SYS.CATWALK' must be inserted within the code section of the user overlay.
- Before any call to CATWALK, the user code must call 'LDFCTWLK' (see LIB.CATWALK.LOADCATWALK) to load the CATWALK overlay. The overlay need not be loaded explicitly.
- The system CATWALK overlay also includes the following routines required by CATWALK:

  LIB.CMPDAT
  LIB.COMPARE
  LIB.DB/DBADD,DBCMPR,DBSUB,TSTZERO
  LIB.GETTASK
  UTIL.FM/BNDSCHK,SEPVOL,SQZSCT

CHKBREAK

SYNTAX	CHKBREAK
PURPOSE	Checks whether a break has been detected by CHKINTR, CHKQUIT, or CHKSTOP since SETBREAK or CHKBREAK was called.
FUNCTION RETURN	0 No Break. 1 Break detected.
NOTE	See SETBREAK.

CHKINTR

SYNTAX

CHKINTR

PURPOSE

Checks for interrupt (ESC) or (CTRL-B), and waits for the next character from the task information stream (TIS).

OUTPUT

After the interrupt, CHKINTR reads the next character from the TIS and does one of the following:

Q	Performs a quit (see CHKSTOP).
N	Terminates any EXECUTE file in progress and performs a quit (see CHKSTOP).
K	Performs a quit and returns to O/S command level regardless of any previous LBLSETs.
1	Prints one line and waits.
2	Prints two lines and waits.
.	.
.	.
.	.
9	Prints nine lines and waits.
P	Prints a page and waits.

FUNCTION RETURN

K	Return to O/S.
N	Terminates execute file and Quits.
Q	Quit.
Other	Character from TIS (byte format).

COMMENTS

All characters are cleared from the TIS when the interrupt is detected.

CHKPROT

SYNTAX CHKPROT(PROTGP,ACTMASK)

PURPOSE Validates access to a file or command, given a protection group and desired type of access.

INPUT

PROTGP Protection group to be validated.

ACTMASK Integer mask indicating access:

Bit	(Leftmost bit is 0):
0-7	Reserved.
8	Model.
9	Read.
10	Write.
11	Execute.
12	Change.
13	Delete.
14	Reserved (must be zero).
15	Type of match.
	ON Subset match of access.
	OFF Exact match of access

FUNCTION RETURN

0 Protection group valid for desired access.

C012 Protection group invalid for desired access.

CHKQUIT

SYNTAX

CHKQUIT

PURPOSE

Checks for interrupt (ESC) or (CTRL-B), and waits for the next character from the task information stream (TIS).

OUTPUT

Accesses the next character from the TIS. That character determines the action to be taken:

- Q — Performs a quit (see CHKSTOP).
- N — Terminates any EXECUTE file in progress and performs a quit (see CHKSTOP).
- K — Performs a quit and returns to O/S command level regardless of any previous LBLSETs.

FUNCTION RETURN

- 0 — No break detected.
- K — Returns to O/S level.
- N — Terminates execute file and performs a QUIT.
- Q — Performs a QUIT.
- Other — Character from TIS (byte format).

COMMENTS

CHKQUIT functions like CHKINTR, but only the characters Q, K, and N are valid for determining the action to be taken.

CHKSTOP

SYNTAX — CHKSTOP

PURPOSE — Searches for an interrupt key. If the search is successful, it performs a QUIT and clears the TIS.

COMMENTS — A QUIT does the following:

- If the current command has called SETBREAK, it sets a flag in the task and returns. This flag may be tested by calling CHECKBREAK.
- It calls LBLGO(INTRQUIT).

CHKSUM

SYNTAX — CHKSUM(CNT,BUF)

PURPOSE — Calculates an additive checksum of words in an array.

INPUT — CNT Number of words.
BUF Array.

FUNCTION RETURN — Total bit value of CNT words from the array BUF.

COMMENT — CHKSUM does not check for an overflow

CHKTRACE

SYNTAX — CHKTRACE(IDUMMY)

PURPOSE — Gets status of trace map output.

INPUT — IDUMMY Non-functional argument.

FUNCTION RETURN — Returns two characters:

'ON' Trace printout is enabled.
'OF' Trace printout is disabled.

CHKUPROT

SYNTAX — CHKUPROT(ACCESS)

PURPOSE — Verifies caller's default protection group for a specific type of file access.

INPUT — ACCESS Access mask for verifying protection:

'80' BIT 8 Model
'40' BIT 9 Read
'20' BIT 10 Write
'10' BIT 11 Execute
'8' BIT 12 Change
'4' BIT 13 Delete
'2' BIT 14 Reserved: Must be zero
'1' BIT 15 Type of match:
0 = Exact match of access.
1 = Subset match of access.

FUNCTION RETURN
0 User has desired access.
'C012'X User lacks desired access.

CKPPTD

SYNTAX CKPPTD(SWT)

PURPOSE Checks for PPTDEV assigned to a task.

OUTPUT

SWT	0	Task has PPTDEV.
	-1	Task has no PPTDEV.

CKRPTD

SYNTAX CKRPTD(SWT)

PURPOSE Checks for RPTDEV assigned to a task.

OUTPUT

SWT	0	Task has RPTDEV.
	-1	Task has no RPTDEV.

CLEARCOM

SYNTAX CLEARCOM.

PURPOSE Fills the command buffer with blanks.

OUTPUT Command buffer blanked out.

NOTE The command buffer is an 80-character Global called COMMAND located in SYSOVLY.

CLOSE

SYNTAX CLOSE(CHANNEL,OPTIONS,SPECIFR)

PURPOSE Closes a file. Can also change file size or delete the file entirely.

INPUT CHANNEL File channel number.

OPTIONS(1)

Bit		
		(Leftmost bit is 0.)
0	ON	Bit 0 is ignored unless the file was opened for create. If opened for create, the file is deleted. The previous file will not be superseded.
	OFF	Normal.
1	ON	User attribute values are set to SPECIFR(1-2). Bit 1 is ignored if: • Channel was used to open file for short or regular read. • File was open for create with supersede and bit 0 is ON.
2	ON	Do not update access date.
3	ON	Change file size to value in SPECIFR(3-4). Ignored under same conditions as bit 1.
4	ON	Use creation date from SPECIFR(5-6).
5	ON	Set CHKSUM and filetype words as indicated in SPECIFR(7-8).
6-12		Reserved (must be zero).
13	ON	OPTIONS(3) contains additional option data.
14		Reserved (must be zero).
15	ON	Error FLAG will be returned as a function name in an error condition.
	OFF	ERROR.FLAG will be called in an error condition.

INPUT (Continued)

OPTIONS(3)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero).

SPECIFR (Words)

(1-2)	New attribute value (if OPTIONS(1) bit 1 is on).
(3-4)	New total sector count for file (if OPTIONS(1) bit 3 is on).
(5-6)	New access date (if OPTIONS(1) bit 4 is set).
(7-8)	Set CHKSUM and filetype (if OPTIONS(1) bit 5 set).

FUNCTION RETURN

If OPTIONS(1) bit 15 is set:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

CLRBIT

SYNTAX

CLRBIT(STRING,INDEX)

PURPOSE

Turns off a specific bit in a bit string.

INPUT

STRING	Bit string array.
INDEX	Index of the bit (leftmost bit is 0).

CLRBREAK

SYNTAX CLRBREAK

PURPOSE Clears task break flag.

COMMENT Task break flag is set to interrupt the current task when CHKINTR detects an ESCAPE, a CTRL B or its equivalent (see SETBREAK).

CMBYTF

SYNTAX CMBYTF(CNT,ARRAY1,N1,ARRAY2,N2)

PURPOSE Compares bytes in two arrays (FORTRAN version).

INPUT CNT Byte count.

ARRAY1	First array to be compared.
N1	Starting byte in first array (N1.GE.1).
ARRAY2	Second array to be compared.
N2	Starting byte in second array (N2.GE.1).

FUNCTION RETURN	0	The arrays are identical.
	(ARRAY1(M) – ARRAY2(M))	Location of the first bytes that differ.

COMMENT Byte indices start with 1.

CMBYTT

SYNTAX CMBYTT(CNT,ARRAY1,N1,ARRAY2,N2)

PURPOSE Compares bytes in two arrays (TPL version).

INPUT

CNT	Byte count.
ARRAY1	First array to be compared.
N1	Starting byte in first array (N1.GE.1).
ARRAY2	Second array to be compared.
N2	Starting byte in second array (N2.GE.1)

FUNCTION RETURN

0	The arrays are identical.
(ARRAY1(M)-ARRAY2(M))	Location of the first bytes that differ.

COMMENT Byte indices start with 0.

CMPDAT

SYNTAX CMPDAT(DATE1,DATE2)

PURPOSE Compares two dates.

INPUT

DATE1 DATE2	Two date-time arrays (system date and time format) to be compared.

FUNCTION RETURN

0	Dates are the same.
>0	DATE1 before DATE2.
<0	DATE1 after DATE2.

COMPN

SYNTAX	COMPN(CNT,ARR1,ARR2)	
PURPOSE	Compares words in two arrays.	
INPUT	CNT	Word count.
	ARR1	First array to be compared.
	ARR2	Second array to be compared.
FUNCTION RETURN	>0	ARR1 is greater than ARR2.
	=0	Arrays are identical.
	<0	ARR1 is less than ARR2.
COMMENT	The comparison is arithmetic — words are treated as signed integers.	

COMPNAM

SYNTAX	COMPNAM(NAM1,NAM2)	
PURPOSE	Compares the ASCII values of two 8-BCD-character names.	
INPUT	NAM1	First name (eight BCD chars).
	NAM2	Second name (eight BCD chars).
FUNCTION RETURN	=0	Names have equal ASCII values.
	>0	NAM1 is greater than NAM2.
	<0	NAM1 is less than NAM2.
COMMENT	The two names are "alphabetized" in ASCII order.	

COMPUS

SYNTAX	COMPUS(A,B)	
PURPOSE	Compares two unsigned integers.	
INPUT	A,B	Unsigned (16-bit) integers.
FUNCTION RETURN	0	A equals B.
	1	A is greater than B.
	−1	A is less than B.

COPEN

SYNTAX COPEN(CHANNEL,FILENAME,OPTIONS,SPECIFR,ARRAY)

PURPOSE Opens a non-catalog file for create or create with supersede.

Restrictions

If open for create with supersede, file cannot be write protected, opened for modify, or opened for create.

INPUT FILENAME(1-41)

Full ASCII filename with a period (.) as the delimiter between components and an exclamation mark (!) following the last character. No component may exceed 20 characters.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Returns extended array.
	OFF	Returns normal array.
1	ON	Creates new file unless file exists (with bit 7 ON, new catalogs are created as needed). Generates an error if the file exists.
	OFF	Creates to supersede an existing file; new space will be allocated. An error is generated if the file does not exist.
2	ON	Allocates initial number of file sectors from SPECIFR(1-2)
	OFF	With bit 1 OFF, new file size is taken as original file size. With bit 1 ON, new file size is taken as task default size.
3	ON	Returns error code 'C020'x as function name if contiguous sectors cannot be allocated.
	OFF	Allows non-contiguous sectors to be allocated, if necessary.

INPUT (Continued)

4	ON	Approximate disc position is given in SPECIFR(3). This has precedence over bit 9 (next fit).
	OFF	Disc positioning of a new file is arbitrary or as per bit 9.
5	ON	Generate an EOF error if attempt to write will cause allocation of additional file space (may be over-written by FWRITE command option).
	OFF	Suppress EOF errors.
6	ON	File protection group is given in SPECIFR(4).
	OFF	Assign task default protection to the file.
7	ON	Create necessary new catalogs.
	OFF	Missing catalogs will generate an error (this bit is ignored if bit 1 is OFF).
8	ON	Catalog protection group is given in SPECIFR(5).
	OFF	Assign task default protection to the catalog.
9	ON	Use next fit allocation strategy. Bit 4 has priority over bit 9.
	OFF	Arbitrary or as per bit 4.
10	ON	File type is in SPECIFR(5).
	OFF	Assign type 0 to a newly created file. Under create with supersede, retain the original file type.
11		Reserved (must be zero).
12	ON	If Directories are enabled, use local create catalog.
	OFF	If directories are enabled, use global create catalog.
13	ON	OPTIONS(2) contains additional option data.
	OFF	Ignore OPTIONS(2).
14	ON	Return working directory name count in OPTIONS(3) followed by characters named in OPTIONS (4-42). If no working directory is enabled, value is returned in OPTIONS(3).

INPUT (Continued)

15 ON Error condition results in error FLAG returned as function name.

OFF Error condition results in the call ERROR(FLAG).

OPTIONS(2)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero).

SPECIFR (Word)

(1-2) Total sector count of the new file (unless OPTIONS(1) bit 2 is OFF).

(3) Approximate position on logical file unit (unless OPTIONS(1) bit 4 is OFF). Position is calculated in 1/32s of a file unit.

(4) File protection group (word 13 of file entry — ignored if OPTIONS(1) bit 6 is OFF).

(5) Catalog protection group (ignored if OPTIONS(1) bit 8 is off).

(6) File type (ignored if OPTIONS(1) bit 10 is OFF).

OUTPUT

OPTIONS(1) bit 14 ON:

OPTIONS(3) Directory name character count ≤ if working directory not enabled).

OPTIONS(4-42) Directory name for OPTIONS(3) number of characters.

ARRAY File information array.

OPTIONS(1) bit 0 OFF:

(Words) Normal Array.

(1-2) Disc location always (0,0).

(3-4) Total sector count of file.

OUTPUT (Continued)

OPTIONS(1) bit 0 ON:

(Words)		Extended Array.
(1-2)		Disc location always (0,0).
(3-4)		Total sector count of file.
(5)		Creation date.
(6)		Time of day created (1 second intervals).
(7)		Access date.
(8)		File protection group.
(9)		File status, when bits are ON:
	Bit	(Leftmost bit is 0.)
	0	In use.
	1	Opened for modify in place.
	2	To be deleted when closed if use count is 0 (i.e., old file of create with supersede).
	3	Open for create.
	4	This entry is a temporary one for create with supersede.
	5	Opened for regular read.
	6	Reserved.
	7	File is a catalog.
	8-15	Reserved.
(10)		Number of users currently accessing file or included file, incremented each time the file or included file is opened, decremented each time it is closed.

OUTPUT (Continued)

(11)			Contiguous data flag 0 All data allocated contiguously
	N		Data not contiguous, N is sector count (N 0) of first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).
(12)			User defined attribute value.
(13)			User defined attribute value.
(14)			CHECKSUM.
(15)			File type
	Bit		(Leftmost bit is 0.)
	0-7	=0	Not task protected.
		>0	1 + number of task which has protected the file.
	8-15	0	Not defined.
		1	Catalog.
		2	Object.
		3	Text.
		4	Configuration.
		5	Reserved.
		6	PEP object code.
		7	Core image (overlay).
		8	Command table.
		9	Loader data file (symbol table or library).
		A	Reserved.
		B	Accounting table.
		C-E	Reserved.
		F	Z80 binary files.
		10-1F	Reserved for CGOS.
		20	CADDS 4 part files.
		21	CADDS 4 TVF files.
		22	CADDS 4 figure files.
		23-2F	Reserved for CADDS 4.
		30-AF	Unused.
		B0-BF	Reserved for batch files.
		C0-FE.	Unused.
		FF	Work files (deleted on FMCLEAR).

OUTPUT (Continued)

CHANNEL Channel number assigned to file.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For description of FILE MANAGER ERRORS, refer to SYSNEWS. ERROR.FM.

COPYFILE

SYNTAX

COPYFILE(FTYPE,FCOPY,BUFSIZ,BUF,NAME1,NAME2)

PURPOSE

Copies an FM file.

INPUT

FTYPE = 0 Copy any file (other than a catalog).
≠ 0 Copy only files of type "FTYPE".

FCOPY > 0 Overwrite if file exists.
= 0 Ask user if file exists.
< 0 Calls ERROR if file exists.

BUFSIZ Buffer size in words — must be at least SECTSIZ (usually 256) words long.

BUF File copy buffer area.

NAME1 Name of FM source file.

NAME2 Name of FM destination file.

FUNCTION RETURN

−1	Buffer is too small to copy names.
0	Copy succeeded.
1	No channel available.
2	File is a catalog.
3	Improper type of file.
4	File already exists.

DBADD

SYNTAX	DBADD(ARG1,ARG2,ARG3).	
PURPOSE	Adds two double word integers.	
INPUT	ARG1,ARG2	Double word addends.
OUTPUT	ARG3	Double word sum.

DBCMPR

SYNTAX	DBCMPR(A,B).	
PURPOSE	Compares two unsigned double precision integers.	
INPUT	A,B	Two double precision numbers.
FUNCTION RETURN	>0	A is greater than B.
	=0	A equals B.
	<0	A is less than B.

DBDCR

SYNTAX	DBDCR(ARG)	
PURPOSE	Subtracts one from a double precision integer.	
INPUT	ARG	Double precision integer.
OUTPUT	ARG	Decremented double precision integer.

DBDIV

SYNTAX	DBDIV(DIVIDEND,DIVISOR,REMAINDER).	
PURPOSE	Divides a double precision integer by a single precision integer.	
INPUT	DIVIDEND	Double precision integer (unsigned).
	DIVISOR	Single precision integer (unsigned).
OUTPUT	REMAINDER	Single precision remainder (unsigned).
FUNCTION RETURN	Single precision quotient (unsigned).	
COMMENT	Carry is set if quotient is greater than 'FFFF'x.	

DBHEXLST

SYNTAX DBHEXLST(LIST)

PURPOSE Scans a hex bounds list from the input stream.

OUTPUT LIST Double word integer array containing:

(1) Count of numbers in list (all hex integers).

(2) $XXXX_1$

(3) $YYYY_1$ (or $XXXX_1$, if $YYYY_1$ omitted)

.
.
.

(n*2) $XXXX_n$

(n*2+1) $YYYY_n$ (or $XXXX_n$, if $YYYY_n$ omitted)

COMMENTS

- The text being scanned has the form:

  $XXXX_1$ (-$YYYY_1$) ,...,$XXXX_m$ (-$YYYY_m$)

- INITCHAR and NXTCHAR set up BCD string and system overlay globals for DBHEXLST. The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).

- DBHEXLST will push past blank characters upon termination.

- See HEXLST, INTLST, and HEXNUM.

DBHEXBCD

SYNTAX DBHEXBCD(HEX,BCD)

PURPOSE Converts a double precision integer to a text string using the hexidecimal radix.

INPUT HEX Double precision integer to be converted.

OUTPUT BCD Four-word array (eight characters) containing the converted text string.

DBHXNM

SYNTAX DBHXNM(STRING,HEXVAL)

PURPOSE Converts an ASCII string to a double word hex integer.

INPUT STRING May include 0 through 9 and A through F, with a maximum of 8 characters. CHAR, a system overlay global, contains the first ASCII character. Its type is in CHARTYP.

OUTPUT HEXVAL Two-word hex integer result.

COMMENTS

- INITCHAR and NXTCHAR set up the BCD string and system overlay globals for DBHXNUM. The two routines are automatically called to scan the command line. When any other string of characters is scanned, they must be explicitly called.
- DBHXNM will push past trailing blanks.
- Routines LIB.DBLSH and LIB.DBADD must be loaded.

DBINC

SYNTAX DBINC(ARG)

PURPOSE Adds one to a double precision integer.

INPUT ARG Double precision integer.

OUTPUT ARG Incremented double precision integer.

DBINT

SYNTAX DBINT(RESULT)

PURPOSE Scans the input stream for a double precision decimal integer.

OUTPUT RESULT Positive double precision integer.

COMMENTS

- INITCHAR and NXTCHAR set up the BCD string and system overlay globals for DBINT. The two routines are automatically called to scan the command line. When any other string of characters is scanned, they must be explicitly called.
- DBINT will push past trailing blanks on termination.

DBINTBCD

SYNTAX — DBINTBCD(DBNUM,BCD,COUNT)

PURPOSE — Converts a signed double precision integer to a character string.

INPUT — DBNUM Signed double precision integer.

OUTPUT — BCD BCD character string.

COUNT Length of the string (COUNT≤11).

COMMENT — Character string is left-justified in a BCD array.

DBINTLST

SYNTAX — DBINTLST(LIST,MAX)

PURPOSE — Scans the input stream for a list of double precision integers.

INPUT — MAX Maximum allowable value of LIST (1) (MAX ≤4).

OUTPUT — LIST Bounds list in form:
(1) Number of words in LIST following.
(2-3) Double precision integer X1.
(4-5) Double precision integer Y1 (or X1 if Y1 was omitted in input stream).

COMMENTS

1. The text string being scanned has the form:

   X1 (-Y1) , ..., Xn (-Yn)

2. INITCHAR and NXTCHAR set up character string and system globals for DBINTLST. The two routines are automatically called to scan the command line. When any other stream of characters is scanned, they must be explicitly called.

3. DBINTLST will push past trailing blanks on termination.

DBLE

SYNTAX DBLE(F)

PURPOSE Converts real to double precision real.

INPUT F Single precision real.

FUNCTION RETURN Double precision real.

DBLSH

SYNTAX DBLSH(ARG1,ARG2,ARG3)

PURPOSE Does a double word left shift.

INPUT

ARG1(1-2)	Double word value to be shifted. Zeros will be shifted into low order bits.
ARG2	Shift count 0-31. If 0 or negative, no shift is performed and ARG3 = ARG1.

OUTPUT

ARG3(1-2)	Double word shifted result.

DBMAX

SYNTAX DBMAX(NUM1,NUM2,MAX)

PURPOSE Finds the larger of two double integers.

INPUT
- NUM1 First double integer to be tested.
- NUM2 Second double integer to be tested.

OUTPUT
- MAX Either NUM1 or NUM2, depending on which is larger.

DBMIN

SYNTAX DBMIN(NUM1,NUM2,MIN)

PURPOSE Finds the smaller of two double integers.

INPUT
- NUM1 First double integer to be tested.
- NUM2 Second double integer to be tested.

OUTPUT
- MIN Either NUM1 or NUM2.

DBMUL

SYNTAX DBMUL(ARG1,ARG2,ANSWER)

PURPOSE Multiplies unsigned single word integers, giving double word integer result.

INPUT
- ARG1 Single word integer (Multiplier).
- ARG2 Single word integer (Multiplicand).

OUTPUT
- ANSWER Double word integer.

DBNEG

SYNTAX DBNEG(ARG)

PURPOSE Uses two's complement arithmetic to reverse the sign of a double precision integer.

INPUT
- ARG Signed double precision integer.

OUTPUT
- ARG Two's complement of INPUT argument.

DBRSH

SYNTAX DBRSH(ARG1,ARG2,ARG3)

PURPOSE Executes a double word right shift.

INPUT

ARG1(1-2)	Double word value to be shifted. Zeros will be shifted into high order bits.
ARG2	Shift count 0-31. If 0 or negative, no shift is performed and ARG3 = ARG1.

OUTPUT

ARG3(1-2)	Double word shifted result.

DBSUB

SYNTAX DBSUB(ARG1,ARG2,ARG3)

PURPOSE Subtracts two double word integers.

INPUT

ARG2	To be subtracted from ARG1.

OUTPUT

ARG3	Double word difference, ARG1-ARG2.

DELETE

SYNTAX DELETE(FILENAME,OPTIONS)

PURPOSE Deletes an unused file or catalog.

INPUT

FILENAME — Full ASCII filename with a period as the delimiter between components and an exclamation mark following the last character. Each component of the filename must be 20 characters or less.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0-12		Reserved (must be zero).
13	ON	OPTIONS(2) contains additional option data.
	OFF	OPTIONS(2) ignored.
14	ON	Return working directory name count in OPTIONS(3), followed by name characters in OPTIONS (4-42). Unless a working directory is enabled, value is returned in OPTIONS(3).
15	ON	Error FLAG returned as function name.
	OFF	Error condition results in the call ERROR(FLAG).

OPTIONS(2)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero)

OUTPUT

OPTIONS(1) bit 14 ON:

OPTIONS(3) — Directory name character count ≤ 0 if working directory not enabled).

OPTIONS(4-42) — Directory name for OPTIONS(3) number of chars.

FUNCTION RETURN — OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS — For description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

NOTE — Catalogs and files which are write or access protected cannot be deleted.

DELETEXT

SYNTAX — DELETEXT(FNAME)

PURPOSE — Deletes a text file.

INPUT — FNAME — Filename in FMNAME format:

Word 1	Count of bytes in name (including !).
Word 2-N	Filename (includes ! but does not include &BCD).

FUNCTION RETURN

0	Successful.
Other	File manager error number.

DELFIL

SYNTAX DELFIL(FILENAME,OPTIONS)

PURPOSE Deletes the last catalog level or file specified in the filename. Also deletes any previous catalog levels, provided they contain no other catalogs or files.

INPUT FILENAME(1-41)

Full filename character string with a period (.) as delimiter between components and an exclamation mark (!) following the last character. Each component of the name must be 20 characters or less.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Print all deleted files.
	OFF	Do not print deleted files.
15	ON	If specified catalog level and/or file not found return an 'C055'x or 'C006'x error.
	OFF	If the catalog level or file specified in FILENAME does not exist, then delete all previous catalog levels that contain no subcatalogs or files.

FUNCTION RETURN

<0	FM Error. If file not found, then either 'C006'x or 'C005'x error is returned. A 'C017'x error is returned if the file is in use.
=0	No errors.

DFLOAT

SYNTAX DFLOAT(I)

PURPOSE Converts an integer to a double precision real.

INPUT I Integer.

FUNCTION RETURN Double precision real.

DFLOATL

SYNTAX DFLOATL(L)

PURPOSE Converts a double integer to a double precision real.

INPUT L Double integer.

FUNCTION RETURN Double precision real.

DIVUS

SYNTAX DIVUS(A,B)

PURPOSE Divides unsigned integers.

INPUT A Unsigned integer dividend.

B Unsigned integer divisor.

FUNCTION RETURN Unsigned integer result (A/B).

ELAPTIME

SYNTAX ELAPTIME(ARR,RARR)

PURPOSE Calculates elapsed time.

INPUT

ARR Base time:

- 0 – Year (right two digits).
- 1 – Day of year (starting at one).
- 2 – Minutes of day.
- 3 – Tenths of seconds.

OUTPUT

RARR Elapsed time since base time

- 0 – Year (right two digits).
- 1 – Day of Year.
- 2 – Minutes of day.
- 3 – Tenths of seconds.

FUNCTION RETURN

- 1 Elapsed time is positive.
- −1 Elapsed time is negative.

ERROR

SYNTAX ERROR(VAL)

PURPOSE System error routine.

INPUT VAL Four-digit hex value.

OUTPUT Based on the type of error call, ERROR will store information in various memory locations (hex values):

Absolute Location	*Contents*
18	Memory violation address for: 'FF04'x and 'FF05'x.
19	Error code for 'FF01'x. PC at time of memory violation for: 'FF04'x through 'FF07'x.
1A	Error Flag — Meaning
	'FF01'x — Error.
	'FF02'x — Undefined subroutine.
	'FF03'x — Wild jump.
	'FF04'x — Write violation: Attempt to write read-protected memory.
	'FF05'x — Access violation: Attempt to read or write access protected memory.
	'FF06'x — Instruction violation: Attempt to disable map from task space.
	'FF07'x — Defer violation: Ninth level of indirection exceeded.
1B	Roll table pointer at error call.
1C,1D 1E,1F	AC0,AC1,AC2,AC3 at error call.

NOTE When ERROR is called, its argument, error cause flag, roll table pointer, and index registers are saved. If SYSERLBL has been set and it has positive value, a LBLGO(SYSERLBL) is executed. Otherwise, control is returned to the system.

FILL

SYNTAX FILL(CNT,VALUE,ARR)

PURPOSE Fills an integer array with a given value.

INPUT

CNT	Count of words to fill.
VALUE	The given value to fill them with.
ARR	The array to fill.

OUTPUT

ARR	The filled array.

FILLBYTT

SYNTAX FILLBYTT(CNT,VALUE,ARRAY,N)

PURPOSE Inserts a given value into each slot of a byte array.

INPUT

CNT	Count of bytes to fill.
VALUE	Word containing byte, in byte format.
ARRAY	The byte array to fill.
N	First byte position to be filled ($N \geq 0$).

OUTPUT

ARRAY	Full byte array.

COMMENT Byte indices start at 0.

FILLCHRT

SYNTAX FILLCHRT(CNT,VALUE,ARRAY,N)

PURPOSE Inserts a given value into each slot of a character array.

INPUT

CNT	Count of bytes to fill.
VALUE	Word containing character, in character format.
ARRAY	The character array to fill.
N	First character position to be filled ($N \geq 0$).

OUTPUT

ARRAY	The filled character array.

COMMENT Byte indices start at 0.

FILLDB

SYNTAX FILLDB(CNT,VALUE,ARRAY)

PURPOSE Inserts a double precision integer into each slot of an array.

INPUT

CNT	Number of integers to insert.
VALUE	Double integer value inserted.

OUTPUT

ARRAY	Double integer array filled with COUNT double integer values.

FILLF

SYNTAX FILLF(CNT,VALUE,ARRAY)

PURPOSE Inserts a real value into each slot of an array.

INPUT
- CNT — Number of real numbers to insert.
- VALUE — Real value to insert in array.

OUTPUT
- ARRAY — Real array filled with numbers.

FLBCD

SYNTAX FLBCD (FLT, BCD)

PURPOSE Converts a floating point number to BCD characters.

INPUT
- FLT — Floating point number.

OUTPUT
- BCD — Six-word array containing:
  - (1) Number of BCD characters in the result.
  - (2-6) Left-justified string of BCD characters. There are a maximum of three positions to the right of the decimal point.

FLOATL

SYNTAX FLOATL(L)

PURPOSE Converts double integer to real.

INPUT L Double integer.

FUNCTION RETURN Real.

FMCNTRNM

SYNTAX FMCNTRNM(OLDNAME,NEWNAME)

PURPOSE Removes the second-to-last level of a file management filename.

INPUT OLDNAME Original filename in format:

(1) Number of characters including an exclamation mark(!).

(2-N) Characters in filename, terminated by a period (.).

OUTPUT NEWNAME Contracted filename in same format as OLDNAME.

FUNCTION RETURN

0 Name contracted.

−1 Name too short to contract.

FMEXPNM

SYNTAX FMEXPNM(OLDNAME,ADDNAME,NEWNAME)

PURPOSE Inserts an additional catalog name as the next to last level of the file management filename.

INPUT

	OLDNAME	Original filename in format:
	(1)	Number of characters including a period (.).
	(2-N)	Characters in filename terminated by an exclamation mark (!) (two characters/word).
	ADDNAME	Level to be added in format:
	(1)	Number of characters in level (without punctuation)
	(2-N)	Characters in level (without punctuation — two characters/word).

OUTPUT NEWNAME Resultant filename in same format as OLDNAME.

FMIDAT

SYNTAX

FMIDAT(DATIM)

PURPOSE

Scans date (and time) from input stream, where date (and time) are in the following format:

MM-DD-YY (:HH:MM:SS)

and converts to system date and time format.

OUTPUT

DATIM Two-word date and time array.

FUNCTION RETURN

0	Date and time correct.
−1	Bad date and time.

COMMENTS

- INITCHAR and NXTCHAR set up the BCD string and system overlay globals for FMIDAT. The two routines are automaticlly called to scan the command line. When any other string of characters is scanned, they must be explicitly called.
- FMIDAT will push past blank characters after completing the scan.
- See also, FMTDAT, GETDAT.

FMIDNF

SYNTAX FMIDNF(SUB,LIST,CNT)

PURPOSE Scans a name (up to 20 characters) from the input stream (FORTRAN version).

INPUT

SUB Byte position at which name storage is to begin in array LIST(SUB $\geq$1).

OUTPUT

LIST Name that has been scanned off.

COMMENTS

- INITCHAR and NXTCHAR set up the BCD string and system overlay globals for DBHXNUM. The two routines are automatically called to scan the command line. When any other string of characters is scanned, they must be explicitly called.
- The name scan will terminate on any of the following characters:

  . , / Blank ! \ ^ Rubout

- FMIDNF will push past trailing blanks on termination.
- Byte indices start at 1.
- This routine intended for use with FORTRAN routines.

FMIDNT

SYNTAX FMIDNT(SUB,LIST,CNT)

PURPOSE Scans a name (up to 20 characters) from the input stream.

INPUT

SUB Byte position at which name storage is to begin in array LIST (SUB ≥0).

OUTPUT

LIST Name that has been scanned off.

NOTES

- INITCHAR and NXTCHAR must be called to set up the string scan. The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- The name scan will terminate on any of the following characters:

  . , / Blank ! [] \ ^ Rubout

- FMIDNT will push past trailing blanks on termination.
- Byte indices start at 0.
- This routine is intended for use with TPL routines.

FMNAME

SYNTAX FMNAME(LIST)

PURPOSE Scans a catalog name from the input stream.

OUTPUT LIST Catalog name as follows:

Word 1	Number of bytes in name (including "!").
Word 2-n	Catalog name with a period(.) between each component and a "!" following the last character.

COMMENT INITCHAR and NXTCHAR initialize the scan for FMNAME. The two routines are automatically called to scan the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).

FMNLST

SYNTAX FMNLST(LIST)

PURPOSE Scans the input stream for a list of comma-separated names. The list is defined by FMIDNT.

OUTPUT LIST List of names in the following format:

Word 1	Number of names.
Word 2-n	M + 1 bytes for each name as follows:

Byte 0:	Number of characters in name.
Bytes:	Characters in name.

COMMENTS

- ITCHAR and NXTCHAR initialize the scan for FMNLST. When the command line is scanned, the two routines are automatically called. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- FMNLST is part of a package that includes INITNM and NXTNAM. Call FMNLST to get the list of names, and then call INITNM and NXTAM to get individual names from the list.

FMTDAT

SYNTAX FMTDAT(INSWT,ARRAY,DATE,TIME)

PURPOSE Converts a date from system date and time format (as found in file entries) to printable format.

INPUT

INSWT	Bit		(Leftmost bit is 0.)
	14	ON	Convert seconds (if bit 15 is on).
		OFF	Do not convert seconds.
	15	ON	Convert time.
		OFF	Do not convert time.
ARRAY(1)	Bit		(Leftmost bit is 0.)
	0		AM/PM indicator: ON for PM, OFF for AM.
	1-6		Number of years since 1960 (0 to 63).
	7-10		Month (1-12).
	11-15		Day (1-31).
ARRAY(2)	Time:		Time of day in 1 second intervals (treated as an unsigned number). Only required if INSWT, bit 15 is ON.

OUTPUT

DATE Four-word array of ASCII characters in the following format:

mm-dd-yy

TIME ASCII array of either three (INSWT bit 14 off) or five (INSWT bit 15 on) words in format:

bhh:mm:ssb

The last two words are omitted if INSWT bit 14 is off (b = blank).

NOTE See also, FMIDAT, GETDAT.

FNDVAL

SYNTAX FNDVAL(VALUE,SIZE,ARRAY)

PURPOSE Searches an array for the integer value specified.

INPUT

VALUE	Value to search for.
SIZE	Number of elements in array (≥ 0).
ARRAY	Array of integers to be searched.

FUNCTION RETURN

$=0$	VALUE not in array.
>0	Array index for location of integer equal to VALUE.
<0	Illegal, should not be returned.

FREAD

SYNTAX FREAD(CHANNEL,DLOC,COUNT,BUF,OPTIONS)

PURPOSE Reads an open file or catalog.

INPUT

CHANNEL — Channel number assigned to the file.

DLOC — Two word relative disc location into file.

COUNT — Number of machine words to read (single word).

BUF — Array of words to read.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0		Reserved (must be zero).
1	ON	Do not wait until read is done, return immediately. Request an I/O flag, address will be returned in OPTIONS(2).
	OFF	Wait until read is done before returning (normal).
2	ON	Return actual count of words read in OPTIONS(3)
3-14		Reserved (must be zero).
15	ON	Error condition results in error FLAG returned as function name.
	OFF	Error condition results in the call ERROR(FLAG).

OUTPUT

OPTIONS(2) — I/O flag address if OPTIONS(1) Bit 1 is on Must call TESTIO to test and release the I/O flag.

OPTIONS(3) — Actual count of words read if OPTIONS(1) bit 2 is ON.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

COMMENTS

- The file or catalog must be open for read, modify, or create.
- If the file is not contiguous, the requested disc read may result in several physical disc reads. If so, the function return will not be immediate. Instead, the value will be returned after the last physical disc read is initiated.
- The size of OPTIONS is determined by the bit pattern of OPTIONS(1). If no values are to be returned in OPTIONS(2-3) the argument OPTIONS need only be one word long.

FREEPG

SYNTAX

FREEPG(SPID)

PURPOSE

Frees memory pages.

RESTRICTIONS:

Memory pages can only be freed under the following conditions.

- Only the owner can free pages.
- They must not be in address space.
- No other users may be using them.
- No I/O may be affecting the pages.

INPUT

SPID	System Page ID of the page to be deallocated.

FUNCTION RETURN

0	No errors.
E020	Illegal system page identifier (SPID).
E024	Calling task does not own the page specified by SPID.
E070	The page specified by SPID is not in the callers page frame.

FWRITE

SYNTAX FWRITE(CHANNEL,DLOC,COUNT,BUF,OPTIONS)

PURPOSE Writes to a file that is open for modify or create.

INPUT

CHANNEL — Channel number assigned to the file.

DLOC — Two word relative disc location into file.

COUNT(1) — Number of machine words to write (single word).

COUNT(2) — Minimum number of sectors to be allocated on an expanded write.

BUF — Array of words to write.

OPTIONS(1)

Bit		(Leftmost Bit is 0.)
0	ON	Upon EOF condition, allocate any additional space as needed. Additional space may not be contiguous. Continue writing to the disc.
	OFF	Any attempt to write past EOF generates an error condition.
1	ON	Do not wait until write is done — return immediately. Request an I/O flag, with address to be returned in OPTIONS(2).
	OFF	Wait until write is done before returning (normal).
2	ON	Return actual count of words written in OPTIONS(3).
3	ON	On EOF condition, allocate at least the number of sectors specified in COUNT(2).
	OFF	On EOF condition, allocate exactly the additional space necessary to complete the write request.
	Note	Bit 1 must be ON for bit 3 to be valid.

INPUT (Continued)

4-14		Reserved (must be zero).
15	ON	Error condition results in error FLAG returned as function name.
	OFF	Error condition results in the call ERROR(FLAG)

OUTPUT

OPTIONS(1) bit 1 ON:

OPTIONS(2)	I/O flag address TESTIO must be called to test and release the I/O flag.
OPTIONS(3)	Returns the count of words written.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

NOTE

If the file is not contiguous, the requested disc write may result in several physical disc writes. The function value will not be returned until after the last physical disc write is initiated.

GETBIT

SYNTAX GETBIT(STRING,INDEX)

PURPOSE Returns the value of a given bit.

INPUT
- STRING — Bit string array.
- INDEX — Bit position in STRING (Leftmost bit is 0).

FUNCTION RETURN
- 0 — Bit is OFF.
- 1 — Bit is ON.

GETDAT

SYNTAX GETDAT(ARRAY)

PURPOSE Gives the current date and time in system date and time format.

OUTPUT ARRA — Two-word date and time array:

Word 1

Bit	(Leftmost bit is 0.)
0	AM/PM indicator: ON for PM, OFF for AM.
1-6	Number of years since 1960 (0 to 63).
7-10	Month (1-12).
11-15	Day (1-31).

Word 2 — Time of day in 1 second intervals (interpreted as an unsigned integer)

NOTE See also, FMIDAT, FMTDAT.

THE GETFILE UTILITY PACKAGE

The GETFILE utility package is used to access text files on a line-by-line basis. The normal sequence for reading from a text file would be to open with GETFILE, read lines of text with GETLINE or GETLINEB, and close with GETCLOSE. There are also four routines for manipulating the sequential access pointer: GETCLEAR, GETMARK, GETPOS, and GETSTART.

GETFILE

SYNTAX

GETFILE(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE

Opens a text file for input on a line-by-line basis.

INPUT

FSB — File Status Block (eight words, see below) FSB(1) MUST be set to 1.

BUF — Output Buffer.

BUFSIZ — Length of buffer in sectors.

FNAME — Filename (without the &BCD) in the form returned by FMNAME:

Word 1	Character count (includes '!').
Word 2-n	Filename (includes '!').

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended array in OPTIONS (2-16).
1	ON	FNAME (42-44) contains the fast lookup information returned from CATWALK.
2-13		Reserved (must be xero).
14	ON	Return normal array in OPTIONS (2-5).
15	ON	Error condition results in error FLAG returned as a function name.
	OFF	Error condition results in the call ERROR(FLAG).

OUTPUT (Continued)

OPTIONS(1) bit 14 ON:

(Word)		Normal Array.
(2-3)		Disc location, always (0,0).
(4-5)		Total sector count of file.

OPTIONS(1) bit 0 ON:

(Word)		Extended Array.
(2-3)		Disc location, always (0,0).
(4-5)		Total sector count of file.
(6)		Creation date.
(7)		Time of day of creation in one-second intervals.
(8)		Access date.
(9)		File protection group.
	Bit	(Leftmost bit is 0.)
	0-7	Reserved.
	8-15	File protection group.
(10)		File status, with bit ON:
	Bit	(Leftmost bit is 0.)
	0	In use.
	1	Opened for modify in place.
	2	To be deleted when closed if use count is 0 (i.e., old file of create with supersede).
	3	Opened for create.
	4	Temporary entry for create with supersede.
	5	Opened for regular read.
	6	Reserved.

OUTPUT (Continued)

	7		File is a catalog.
	8		File is a volume.
	9-15		Reserved.
(11)			Number of users currently accessing file or included file — incremented each time the file or included file is opened, decremented each time it is closed.
(12)			Contiguous data flag: 0 All data allocated contiguously. N Data not contiguous, N is sector count (N 0) of first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).
(13)			User-defined attribute value.
(14)			User-defined attribute value.
(15)			CHKSUM.
(16)			File type
	Bit		(Leftmost bit is 0.)
	0-7	=0	Not task protected.
		>0	1 + number of task which has protected the file.
	8-15	0	Not defined.
		1	Catalog.
		2	Object.
		3	Text.
		4	Configuration.
		5	Reserved.
		6	PEP object code.
		7	Core image (overlay).
		8	Command table
		9	Loader data file (symbol table or library).
		A	Reserved.
		B	Accounting table.
		C-E	Reserved.
		F	Z80 binary files.

OUTPUT (Continued)

10-1F	Reserved for CGOS.
20	CADDS 4 part files.
21	CADDS 4 TVF files.
22	CADDS 4 figure files.
23-2	Reserved for CADDS 4.
30-AF	Unused.
BO-BF	Reserved for batch files.
CO-FE	Unused.
FF	Work files (detected on FMCLEAR).

Description of FSB (File Status Block)

FSB(1) Status:

1. No file open.
2. Reading file.
3. At end-of-file.
4. Reading without BUF in memory.
5. At EOF without BUF in memory.

FSB(2)	Input channel number.
FSB(3-4)	Relative DLOC in file.
FSB(5-6)	Bytes remaining in file.
FSB(7)	Byte pointer.
FSB(8)	Size of user's disc buffer in words.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

Refer to ERROR. For description of FILE MANAGER ERRORS.

NOTES

- Routines in the GETFILE family have no local variables to maintain. Only the integrity of FSB and BUF is important.
- BUFSIZ must be a multiple of SECTSIZ that is less than 32K bytes (16384 words).

GETLINE

SYNTAX	GETLINE(FSB,BUF,MAX,LEN,LINE)	
PURPOSE	Retrieves next line from text file opened by GETFILE.	
INPUT	FSB	File Status Block.
	BU	Disc buffer.
	MAX	Length of line buffer in characters.
OUTPUT	LEN	Length of the line (-1 if at END OF FILE).
	LINE	Character string array of LEN characters.
ERRORS	For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.	
NOTE	MAX must be greater than or equal to the longest line in the file, otherwise an error will be triggered.	

GETLINEB

SYNTAX

GETLINEB(FSB,BUF,MAX,LEN,LINE,OFFSET)

PURPOSE

Retrieves next line from text file opened by GETFILE. This routine is equivalent to GETLINE with the addition of a byte offset.

INPUT

FSB	File Status Block.
BUF	Disc buffer.
MAX	Length of line buffer in characters.
OFFSET	Byte offset in LINE (counted from 0) for the contents of BUF.

OUTPUT

LEN	Length of the line (-1 if at END OF FILE).
LINE	Character string array of LEN characters.

NOTES

- Complete documentation, including a full description of the FSB (File Status Block), may be found under GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.
- MAX must be greater than or equal to the longest line in the file, otherwise an error will be triggered.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

GETCLEAR

SYNTAX GETCLEAR(FSB)

PURPOSE Marks the FSB (File Status Block) of an open file to indicate that the text file buffer is no longer available and must be loaded into memory the next time that the routine GETLINE is called.

INPUT FSB File Status Block.

NOTE
- Additional documentation, including a description of the FSB (File Status Block), may be found under LIB.FM.GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

GETPOS

SYNTAX GETPOS(FSB,POS)

PURPOSE Used with textfiles, restores a position marked by GETMARK.

INPUT FSB File Status Block.

OUTPUT POS(7) Saved position from GETMARK.

NOTES
- Complete documentation, including a full description of the FSB (File Status Block), may be found under GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

GETMARK

SYNTAX GETMARK(FSB,POS)

PURPOSE Marks the current position in a text file being read by GETFILE.

INPUT FSB File Status Block.

OUTPUT POS(7) Saved position information.

NOTES
- Complete documentation, including a full description of the FSB (File Status Block), may be found under GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

GETSTART

SYNTAX GETSTART(FSB,BUF)

PURPOSE Moves the pointer to the top of a text file opened by GETFILE.

INPUT FSB File Status Block.

BUF Disc buffer.

NOTES
- Complete documentation, including a full description of the FSB (File Status Block), may be found under GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

GETCLOSE

SYNTAX GETCLOSE(FSB,BUF)

PURPOSE Closes the text file opened by the routine GETFILE.

INPUT FSB File Status Block.

BUF Disc buffer.

NOTES
- Additional documentation, including a description of the FSB (File Status Block), may be found under LIB.FM.GETFILE.
- This routine must be used in conjunction with LIB.FM.GETFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

GETFLD

SYNTAX — GETFLD(STRING,FBIT,SIZE)

PURPOSE — Retrieves the value in a field and returns it as a right-justified integer.

INPUT

STRING — Bit string containing the field to be returned.

FBIT — The location of the field, specified by the displacement in bits between the high-order (leftmost) bit of the field and the high-order bit of string.

SIZE — The size of the field in bits (0 < SIZE < 17).

FUNCTION RETURN — An integer value for the contents of the field.
More exactly:

```
FOR I = 1,SIZE.
RETURN (15-SIZE+I)  STRING(FBIT-1+I).
```

RETURN and STRING are treated as bit strings indexed from 0.

ERRORS — If there is an error in the argument, GETFLD calls ERROR.

NOTE — LIB.DBLSH must also be loaded. See also TSTBIT, SETBIT, CLRBIT, PUTBIT, PUTFLD, and GETBIT.

GETLABEL

SYNTAX — GETLABEL(TBLABEL,SAVLABEL)

PURPOSE — Accesses a system label from the TASKBLOCK and saves it in: SAVLABEL.

INPUT — TBLABEL — Identifies the desired system label from the TASKBLOCK:

- −1 TAPERLBL
- −2 SYSERLBL
- −3 INTRQUIT

OUTPUT — SAVLABEL — Destination of the label (2-word array).

NOTES

- For safety, GETLABEL should only be used in user address space.
- System label values and references are the same as LBLSET and LBLGO.

ERRORS — 'F00D'x — TBLABEL specified is illegal (bad arg).

GETPG

SYNTAX GETPG(BUFFER,SPID,PERMIT)

PURPOSE Reads a page slot of a user's page frame, returning the:

- SPID of the system memory page in the page slot.
- Permission associated with the slot.

INPUT

BUFFER A memory buffer, one memory page long ('800'X words) that is aligned on a page boundary.

OUTPUT

SPID System Page ID of the page in the page slot specified by SLOTID. If there is no page in the slot, SPID will be NULL.

PERMIT User access for the page slot identified by SLOTID:

P&RW	Any access is legal.
P&RO	Read-only.
P&NO	No legal access.

FUNCTION RETURN

0	No errors.
E010	BUFFER is not aligned on page boundary.

NOTE The symbolic constants are defined in SYM.EQU.MEMMAN and SYM.EQUF.MEMMAN.

GETPGP

SYNTAX GETPGP(SPID,PROT)

PURPOSE Reads the parameters of a system page.

INPUT SPID System page ID of a memory page.

OUTPUT PROT Protection attributes of this page:

P&RW	Any access is legal.
P&RO	Read access only.
P&NO	No legal access.

FUNCTION RETURN

0	No errors.
E020	Illegal system page identifier (SPID).

NOTE The symbolic constants are defined in SYN.,EQU.MEMMAN and SYM.EQUF.MEMMAN.

GETPPTD

SYNTAX

GETPPTD(ARG)

PURPOSE

Attaches punch paper tape unit (PPTDEV) to a task.

OUTPUT

ARG Set as follows:

If PPTDEV is already attached, ARG = 0.

If no PPTDEV is attached, prompts for a task unit name, unique name, or unit type to be typed in at the COMDEV:

- No name typed in, ARG = −1.
- Name typed in:

Two-character name, attempts to declare unit as PPTDEV.

Four-character name, attempts to attach a unit with this unique name, or of this type as 'PP', and attach it as PPTDEV

If attaching or declaration is unsuccessful, asks for another name. Otherwise ARG = 0.

FUNCTION RETURN

ARG

GETPUNCH

SYNTAX GETPUNCH(VALUE)

PURPOSE Gets the parity option for the PPTDEV.

OUTPUT VALUE parity option for task PPTDEV, where:

0: no parity, transparent data transfer.
1: odd parity.
2: even parity.
3: marked parity, MSB of byte set to 1.
4: spaced parity, MSB of byte set to 0.

FUNCTION RETURN	0	No error detected.
	−1	No PPTDEV for task.
	OTHER	Error return from GETDEV routine.

GETRPTD

SYNTAX GETRPTD(ARG)

PURPOSE Attaches read paper tape unit (RPTDEV) to a task.

OUTPUT ARG Set as follows:

If RPTDEV is already attached, ARG = 0.

If no RPTDEV is attached, prompts for a task unit name, unique name, or unit type to be typed in at the COMDEV:

- No name typed in, ARG = −1.
- Name typed in:

  Two-character name, attempts to declare unit as RPTDEV.

  Four-character name, attempts to attach a unit with this unique name, or of this type as 'RP', and attach it as RPTDEV.

If attaching or declaration is unsuccessful, asks for another name. Otherwise ARG = 0.

FUNCTION RETURN ARG

GETSTAT

SYNTAX GETSTAT(TYP,TYPID,TIMBLK)

PURPOSE Reads task or system statistics blocks.

INPUT

TYP — Type of statistics block to return:

=1, System statistics block (SYSSTAT, 4 Words).
=2, Task statistics block (TASKSTAT, 4 words).
=3, Total CPU time used by logged out tasks (2 words).

TYPID — Contains address of taskblock when TYPE=2.

Type	*TSKADDR*	
=2	=0	Return task CPU usage.
=2	≠0	TSKADDR taskblock pointer.
≠2	Ignored	

OUTPUT

TIMBLK — Double-word array to receive either task or system statistics block.

GETTASK

SYNTAX	GETTASK
PURPOSE	Returns task number.
FUNCTION RETURN	Task number (≥ 0).
NOTE	GETTASK may only be called from TPL routines. For FORTRAN version, see GETTASKF.

GETTASKF

SYNTAX	GETTASKF(INUM)
PURPOSE	Returns task number (FORTRAN version).
OUTPUT	INUM The task number.
FUNCTION RETURN	The task number.

GLP

SYNTAX GLP(DEVICE,ANTECC,CNT,LINE,POSTCC)

PURPOSE Outputs a line of characters to device(s).

INPUT

DEVICE Device Selector:

0	Use devices and/or hardfile selected by the toggles.
'CM'	COMDEV only (need not be toggled on).
'C+'	Use the COMDEV plus any other devices and/or hardfile selected by the toggles.
'HD'	HARDEV/hardfile only (must be toggled on).
'LP'	Use physical HARDEV with implicit page eject. The task must have a device declared as the task HARDEV.

ANTECC Carriage control before the line is printed:

'11'x	Do page eject before the line.
0	Print on the current line.
Other	Number of lines to skip before printing.

CNT Number of characters in the line.

LINE Characters to output (will be followed by a carriage return).

POSTCC Carriage control after the line is printed:

'11'	Do page eject after the line.
0	No carriage advance (allows overprinting).
Other	Number of lines to skip after printing.

HEAPSORT

SYNTAX HEAPSORT(ARRAY,N,COMPFUN)

PURPOSE Sorts an internal file of single word records. A user-supplied function compares keys.

INPUT

ARRAY	Internal file of single word records.
N	Number of records in the file.
COMPFUN	Record comparison function with a return that depends on the relation of KEY1 and KEY2:

KEY1	*COMPFUN*
< KEY2	< 0
= KEY2	= 0
> KEY2	> 0

OUTPUT ARRAY Sorted internal file.

NOTE The time required for HEAPSORT is O(N log N).

HEXBCD

SYNTAX HEXBCD(HEX,BCD)

PURPOSE Converts a hex word to BCD.

INPUT HEX Hex word to be converted.

OUTPUT BCD Two-word array containing four characters.

HEXDMP

SYNTAX HEXDMP(LOC1,LOC2,BIAS,BUF)

PURPOSE Creates hex BCD dump of an array.

INPUT

LOC1 Lower hex bound subscript to appear on dump.
LOC2 Upper hex bound subscript to appear on dump.

BIAS Hex subscript corresponding to first word of BUF (same units as LOC1 and LOC2). BIAS can be used to dump portions of core before BUF without knowing their address.

BUF Array containing information to be dumped.

NOTE Each line (between LOC1 and LOC2) contains start address for the line, and eight hex dump words. Core constants appear as .... . Lines identical to the previous line will not be printed. Omitted lines have .... in the address field. One line skipped at the end of HEXDMP output.

EXAMPLE To dump '100'x words starting at array Q, use:

```
CALL HEXDMP(0,'FF'x,0,Q)
```

To dump '10'x words just before array Q, use:

```
CALL HEXDMP(0,'10'x,'10'x,Q)
```

HEXLST

SYNTAX HEXLST(LIST)

PURPOSE Scans the input stream for a hex bounds list.

OUTPUT

LIST(1)	Count of pairs in list (all hex integers)
LIST(2)	$XXXX_1$
LIST(3)	$YYYY_1$ (or $XXXX_1$, if $YYYY_1$ omitted)
.	
.	
.	
LIST(n*2)	$XXXX_n$
LIST(n*2+1)	$YYYY_n$ (or $XXXX_n$, if $YYYY_n$ omitted)

NOTES

- The text being scanned has the form:

  $$XXXX_1\ (-YYYY_1\ ),\ldots,XXXX_m\ (-YYYY_m\ )$$

- INITCHAR and NXTCHAR set up BCD string and system overlay globals for HEXLST. The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream characters (i.e., a line input with TYPIN).

- HEXLST will push past blank characters upon termination.

- See INTLST and HEXNUM.

HEXNUM

SYNTAX HEXNUM(HEXVAL)

PURPOSE Scans a hex number from the input stream.

OUTPUT HEXVAL Hex integer result.

NOTES
- The hex number must fit in a 16 bit integer.
- INITCHAR and NXTCHAR set up BCD string and system overlay globals for HEXNUM. The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- HEXNUM will push past blank characters upon termination.

HIBERN8

SYNTAX HIBERN8(TIMER)

PURPOSE Deactivates a task for a specified time period.

INPUT TIMER Number of clock ticks (10 ms/tick) task is to be deactivated.

NOTE The task is reactivated when the timer reaches zero. If the routine is called with an argument of zero or less, then the task will be deactivated for (32768 + TIMER) + 32767 clock ticks.

IDENT

SYNTAX — IDENT(NAME)

PURPOSE — Scans the input stream for an eight-BCD-character identifier beginning with an alphabetic character.

OUTPUT

NAME	Eight-BCD-character identifier,left-justified, and padded with blanks.

ERRORS

'801'x	First character is not an & or an alphabetic character.
'802'x	More than eight characters in identifier.

NOTES

- INITCHAR and NXTCHAR set up BCD string and system overlay globals for IDENT. The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- IDENT will push past blank characters upon termination.
- The scan terminates when a special character is found.

IFIXD

SYNTAX — IFIXD(D)

PURPOSE — Converts a double precision real to a single-word integer.

INPUT

D	Double precision real number.

FUNCTION RETURN — Single-word integer.

INITCHAR

SYNTAX INITCHAR(ARRAY,N)

PURPOSE Initializes array and character count for NXTCHAR.

INPUT
- ARRAY — Array of characters.
- N — Count of characters in array.

NOTES
- INITCHAR and NXTCHAR must be called in preparation for all scan routines.
- INITCHAR and NXTCHAR are automatically called to set up scanning of the command line. The two routines must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- Initializes character accessing function NXTCHAR for use on ARRAY with N characters. NXTCHAR sets up system overlay globals.

INITNM

SYNTAX INITNM(NMLST,NINFO)

PURPOSE Initializes scanning of the input stream for list of comma-separated names. The characteristics of a name are defined by FMIDNT.

INPUT

NMLST: FM NAME LIST of the form:

Word 1:	Count of names.	
Word 2-n:	M+ Bytes for each name as follows:	
	BYTE 0:	Count of characters in name.
	BYTES 1-M:	Characters in name.

NINFO: Two-word array for scanning name list.

COMMENT INITNM is part of a package that includes FMNLST and NXTNAM. Call FMNLST to get the list of names, and then call INITNM and NXTNAM to get individual names from the list.

INSESORT

SYNTAX INSESORT(ARRAY,N,COMPFUN)

PURPOSE Sorts an internal file of single word records. Keys are compared by a user-supplied function.

INPUT

ARRAY	Internal file of single word records.
N	Number of records in the file.
COMPFUN	Record comparison function with a return that depends on the relation of KEY1 and KEY2:

KEY1	*COMPFUN*
< KEY2	< 0
= KEY2	= 0
> KEY2	> 0

OUTPUT ARRAY Sorted internal file.

NOTE INSESORT uses a straight insertion sort. The time required is 0(N). This is a stable sort.

INSESRTN

SYNTAX INSESRTN(RECS,NREC,COMPFUN,RECLEN,REC)

PURPOSE Sorts records using a straight insertion sort.

INPUT

RECS — Array containing data to be sorted.

NREC — Number of records to be sorted.

COMPFUN — Record comparison function with a return that depends on the relation of KEY1 to KEY2:

KEY1	*COMPFUN*
< KEY2	< 0
= KEY2	= 0
> KEY2	> 0

RECLEN — Length of record, in words.

REC — A caller supplied scratch, RECLEN words long.

OUTPUT Sorted array.

NOTES

- INSESRTN is a straight insertion sort that follows the Knuth Algorithm *(Knuth, Vol. 3, p. 81)*. Comments of the form, SN, refer to steps in the Knuth Algorithm.
- SHELSRTN, a shell sort, is generally preferable to INSESRTN unless a stable sort is required. It is significantly faster than INSESRTN when more than 25 records are sorted. The two routines have similar amounts of code.

INT

SYNTAX INT(RESULT)

PURPOSE Scans a decimal integer from the input stream.

OUTPUT RESULT Positive or negative decimal integer.

NOTES
- INITCHAR and NXTCHAR set up character string and system globals for INT.
- The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e. a line input with TYPIN).
- INT will push past blank characters upon termination.

INTBCD

SYNTAX INTBCD(NUM,CHARS)

PURPOSE Converts an integer to a character string.

INPUT NUM Integer to be converted.

OUTPUT CHARS Three-word array containing character string.

NOTES
- Character string is right-justified in CHARS.
- Unfilled characters are padded with blanks.
- A minus sign, if needed, is inserted in front of the leftmost character.

INTLST

SYNTAX INTLST(LIST)

PURPOSE Scans the input stream for a decimal bounds list.

OUTPUT

LIST(1)	Count of pairs in list (all hex integers).
LIST(2)	X_1
LIST(3)	Y_1 (or X_1, if Y_1 omitted).
.	
.	
.	
LIST(n*2)	X_n
LIST(n*2+1)	Y_n (or X_n, if Y_n omitted).

COMMENTS

- The text string being scanned has the form:

  $X_1(-Y_1)$,...,$X_m(-Y_m)$

- INITCHAR and NXTCHAR set up character string and system globals for INTLST.
- The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).
- INTLST will push past blank characters upon termination.

IPDL

SYNTAX IPDL(P,L)

PURPOSE Initialize a push down list.

INPUT
- P Array to be a push-down list.
- L Length of the P array.

OUTPUT
- P Push-down list.

NOTE Using IPDL allows the following routines to use the push-down list:

POP
POPN
PUSH
PUSHN

ISNGL

SYNTAX ISNGL(L)

PURPOSE Converts a double integer to an integer.

INPUT
- L Double integer.

FUNCTION RETURN Integer.

LABS

SYNTAX LABS(I)

PURPOSE Computes the absolute value of a double integer.

INPUT I Double integer.

FUNCTION RETURN Absolute value of I.

LBLGO

SYNTAX LBLGO(ARG)

PURPOSE Transfers control to label set in LBLSET.

INPUT

ARG-1	TAPERLBL to be used.
ARG-2	SYSERLBL to be used.
ARG-3	INTRQUIT to be used.
Other	Two-word label array containing information set in LBLSET.

ERRORS 'F00D'x ARG(1) bad (is core constant or ≤0).

NOTES

- On a LBLGO, the roll table pointer is set to the value in ARG(2), and ARG(1) is put into return address location of that roll table entry, a RETURN is then executed.
- See also, GETLABEL, PUTLABEL, LBLSET, and RESETLBL.

LBLSET

SYNTAX LBLSET(ARG,LBL)

PURPOSE Sets up information for LBLGO transfer.

INPUT

LBL	Statement label in calling code to which control is to be transferred if LBLGO.(ARG) is called.
ARG-1	TAPERLBL to be set.
ARG-2	SYSERLBL to be set.
ARG-3	INTRQUIT to be set.
Other	ARG is two-word label array to be set.

OUTPUT The selected two-word label array is set as:

(1) Address of LBL.
(2) Current roll table pointer.

NOTES

- LBLSET can only be called from TPL because FORTRAN statement labels may not be passed as arguments.
- The following calls will reset the system labels to default operation:

```
CALL RESETLBL(-1)
CALL RESETLBL(-2)
CALL RESETLBL(-3)
```

- All system labels are set to the default values at the start of every command.

LDBYTF

SYNTAX LDBYTF(ARRAY,N)

PURPOSE Loads a byte (FORTRAN version).

INPUT
ARRAY Array containing bytes.
N Byte position (N≥ 1).

FUNCTION RETURN The Nth byte of ARRAY in byte format.

COMMENT Byte indices start at 1.

LDBYTT

SYNTAX LDBYTT(ARRAY,N)

PURPOSE Loads a byte.

INPUT
ARRAY Byte array.
N Byte position (N≥ 0)

FUNCTION RETURN The Nth byte of ARRAY in byte format.

COMMENT Byte indices start at 0.

LDCHRF

SYNTAX	LDCHRF(ARRAY,N)
PURPOSE	Loads a character (FORTRAN version).
INPUT	ARRAY Array containing bytes. N Byte position (N≥ 1).
FUNCTION RETURN	The Nth byte of ARRAY in character format.
COMMENT	Byte indices start at 1.

LDCHRT

SYNTAX	LDCHRT(ARRAY,N)
PURPOSE	Loads a character (TPL version).
INPUT	ARRAY Array containing bytes. N Byte position (N≥ 0).
FUNCTION RETURN	The Nth byte of ARRAY in character format.
COMMENT	Byte indices start at 0.

LFIX

SYNTAX	LFIX(F)
PURPOSE	Converts a real number to a double integer.
INPUT	F Real.
FUNCTION RETURN	Double integer.

LFIXD

SYNTAX	LFIXD(D)
PURPOSE	Converts a double precision real to a double integer.
INPUT	D Double precision real.
FUNCTION RETURN	Double integer.

LP

SYNTAX LP(CARRCONT,ARGCNT,LINE)

PURPOSE Outputs a line of characters to COMDEV (and HARDEV if hard copy toggle on).

INPUT

CARRCONT Carriage control.

='0-F'x	Skip CARRCONT lines before outputting characters.
='11'x	Eject page before outputting characters

ARGCNT Number of characters to be output.

LINE Packed character string.

NOTES

- The system routine CHKINTR is called when a line of characters is output by LP. This may result in characters being removed from the TIS (see CHKINTR).
- See also TYPE, TYPOUT.

MAXIMUM

SYNTAX — MAXIMUM(I,J)

PURPOSE — Finds the larger of two unsigned integers.

INPUT
- I Unsigned Integer.
- J Unsigned Integer.

FUNCTION RETURN — Larger of I and J.

MINIMUM

SYNTAX — MINIMUM(I,J)

PURPOSE — Finds the smaller of two unsigned integers.

INPUT
- I Unsigned integer.
- J Unsigned integer.

FUNCTION RETURN — Smaller of I and J.

MODIFILE

SYNTAX MODIFILE(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE Opens a binary sequential file for modification.

INPUT

FSB File Status Block.

BUF Disc buffer.

BUFFSIZ Size of disc buffer in sectors.

FNAME Filename in FM format:

Word 1	Character count (including !).
Word 2-n	Filename (terminated by !).

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended array in OPTIONS(2-16).
14	ON	Return normal array in OPTIONS(2-5).
15	ON	Function returns errors.
	OFF	Errors are processed by the system error handler.

OUTPUT OPTIONS(2-16) File information array:

OPTIONS(1) bit 0 OFF:

(Word)	Normal Array
(2-3)	Disc location always (0,0).
(4-5)	Total sector count of file.

OPTIONS(1) bit 0 ON:

(Word)	Extended Array
(2-3)	Disc location always (0,0).
(4-5)	Total sector count of file.

OUTPUT (Continued)

(6)		Creation date.
(7)		Creation time, in one-second intervals.
(8)		Access date.
(9)		File protection group:
	Bit	
	0-7	Reserved.
	8-15	File protection group
(10)		File status:
		Status as indicated when bit ON.
	Bit	
	0	File in use.
	1	File opened for modify in place.
	2	File to be deleted when closed if use count is 0 (i.e., old file of create with supersede).
	3	File Open for create.
	4	This entry is a temporary one for create with supersede.
	5	File opened for regular read.
	6	Reserved.
	7	File is a catalog.
	8	File is a volume.
	9-15	Reserved.
(11)		Use count:
		Record of number of users currently accessing file or included file, incremented each time the file or included file is opened, decremented each time the file or included file is closed.

OUTPUT (Continued)

(12)			Contiguous data flag
		0	Contiguous data allocation.
		>0	Data not contiguous, N is sector count of the file's first contiguous chunk of data (if file has 0 length this is indicated by ARRAY(3-4)).
(13)			User defined attribute value.
(14)			User defined attribute value.
(15)			CHKSUM.
(16)			File type
	Bit		(Leftmost bit is 0.)
	0-7	=0	Not task protected.
		>0	1 + number of task which has protected the file.
	8-15	0	Not defined.
		1	Catalog.
		2	Object.
		3	Text.
		4	Configuration.
		5	Reserved.
		6	PEP object code.
		7	Core image (overlay).
		8	Command table.
		9	Loader data file (symbol table or library).
		A	Reserved.
		B	Accounting table.
		C-E	Reserved.
		F	Z80 binary files.
		10-1F	Reserved for CGOS.
		20	CADDS 4 part files.
		21	CADDS 4 TVF files.
		22	CADDS 4 figure files.
		23-2F	Reserved for CADDS 4.
		30-AF	Unused.
		B0-BF	Reserved for batch files.
		C0-FE.	Unused.
		FF	Work files (deleted on FMCLEAR).

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For description of FILE MANAGER ERRORS, refer to SYSNEWS ERROR.FM.

MOPEN

SYNTAX MOPEN(CHANNEL,FILENAME,OPTIONS,ARRAY)

PURPOSE Opens a file for modify in place.

Restriction

Read and write access are needed. File must not be in use.

INPUT FILENAME

(Word)

(1-41) Full ASCII filename with a period (.) as delimiter between components and an exclamation mark (!) following the last character. Each component of the filename must be 20 characters or less.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended array.
	OFF	Return normal array.
1-12		Reserved (must be zero).
13	ON	OPTIONS(2) contains additional option data.
	OFF	OPTIONS(2) ignored.
14	ON	Return working directory name count in OPTIONS(3) followed by characters named in OPTIONS (4-42). If working directory not enabled, value is returned in OPTIONS(3).
15	ON	Error condition results in error FLAG returned as function name.
	OFF	Error condition results in the call ERROR(FLAG).

INPUT (Continued)

OPTIONS(2)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero).

OUTPUT

OPTIONS(1) bit 14 ON:

OPTIONS(3)	Directory name character count, or ≤ 0 if working directory not enabled.
OPTIONS(4-42)	Directory name for OPTIONS(3) number of characters.
ARRAY	File information array:

OPTIONS(1) bit 0 OFF:

(Word)	Normal Array
(1-2)	Disc location always (0,0).
(3-4)	Total sector count of file.

OPTIONS(1) bit 0 ON:

(Word)	Extended Array
(1-2)	Disc location always (0,0).
(3-4)	Total sector count of file.
(5)	Creation date.
(6)	Creation time, in one-second intervals.
(7)	Access date.
(8)	File protection group:

Bit	
0-7	Reserved.
8-15	File protection group.

OUTPUT

(9) File status:

Status as indicated when bit ON.

Bit	
0	File in use.
1	File opened for modify in place.
2	File to be deleted when closed if use count is 0 (i.e., old file of create with supercede).
3	File Open for create.
4	This entry is a temporary one for create with supersede.
5	File opened for regular read.
6	Reserved.
7	File is a catalog.
8	File is a volume.
9-15	Reserved.

(10) Use count:

Record of number of users currently accessing file or included file, incremented each time the file or included file is opened, decremented each time the file or included file is closed.

(11) Contiguous data flag.

0	Contiguous data allocation.
>0	Data not contiguous, N is sector count of the first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).

(12) User defined attribute value.

(13) User defined attribute value.

(14) CHKSUM.

(15) File type

Bit		
		Leftmost bit is 0.
0-7	=0	Not task protected.
	>0	1 + number of task which has protected the file.

OUTPUT (Continued)

8-15	0	Not defined.
	1	Catalog.
	2	Object.
	3	Text.
	4	Configuration.
	5	Reserved.
	6	PEP object code.
	7	Core image (overlay).
	8	Command table.
	9	Loader data file (symbol table or library).
	A	Reserved.
	B	Accounting table.
	C-E	Reserved.
	F	Z80 binary files.
	10-1F	Reserved for CGOS.
	20	CADDS 4 part files.
	21	CADDS 4 TVF files.
	22	CADDS 4 figure files.
	23-2F	Reserved for CADDS 4.
	30-AF	Unused.
	B0-BF	Reserved for batch files.
	C0-FE	Unused.
	FF	Work files (deleted on FMCLEAR).

CHANNEL Channel number assigned to the file.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

MOV

SYNTAX MOV(CNT,ARR1,ARR2)

PURPOSE Moves integer data from one array to another.

INPUT
- CNT — Number of integer words to be moved.
- ARR1 — Array containing integer data.

OUTPUT
- ARR2 — Array filled with data from ARR1.

MOVB

SYNTAX MOVB(CNT,ARR1,ARR2)

PURPOSE Moves integer data from one array to another, starting at the end of the array and proceeding backwards through the array.

INPUT
- CNT — Number of integer words to be moved.
- ARR1 — Last word of source array.

OUTPUT
- ARR2 — Last word of destination array, filled with data from the source array.

MOVD

SYNTAX MOVD(CNT,ARR1,ARR2)

PURPOSE Moves double precision real data from one array to another.

INPUT
- CNT — Number of double real precision numbers to be moved.
- ARR1 — Source array.

OUTPUT
- ARR2 — Destination array, filled with data from the source array.

MOVEWORD

SYNTAX MOVEWORD(ARRAY1,OFF1,CNT1,ARRAY2,OFF2,CNT2,NUM2, EFLAG)

PURPOSE Moves a word (alphabetic string) from one array to another, stopping before any non-alphabetic character. Lower case characters are converted to upper case.

INPUT
- ARRAY1 — Packed array of characters to be moved.
- OFF1 — Byte offset within ARRAY1 to start.
- CNT1 — Total number of characters in ARRAY1.
- ARRAY2 — Destination array.
- OFF2 — Byte offset within ARRAY1 to start.
- CNT2 — Size in bytes of ARRAY2.
- NUM2 — Second double integer to be tested.

OUTPUT
- EFLAG — Error flag:
  - 0 — No errors.
  - ≠0 — Move would overflow destination buffer.

MOVF

SYNTAX MOVF(CNT,ARR1,ARR2)

PURPOSE Moves real data from one array to another.

INPUT
- CNT Number of real numbers to be moved.
- ARR1 Source array.

OUTPUT
- ARR2 Destination array, filled with data from the source array.

MOVL

SYNTAX MOVL(CNT,ARR1,ARR2)

PURPOSE Moves double precision integer data from one array to another.

INPUT
- CNT Number of double integer precision numbers to be moved.
- ARR1 Source array.

OUTPUT
- ARR2 Destination array, filled with data from the source array.

MVBYTF

SYNTAX MVBYTF(CNT,ARRAY1,N1,ARRAY2,N2)

PURPOSE Moves bytes (or characters) from one array to another.

INPUT

CNT	Byte count.
ARRAY1	Array containing bytes to be moved.
N1	First byte position in ARRAY1 to be moved (N1≥ 1).
N2	First byte position in ARRAY2 to receive data (N2≥ 1).

OUTPUT ARRAY2 Destination array.

COMMENTS Byte indices start at 1.

MVBYTT

SYNTAX MVBYTT(CNT,ARRAY1,N1,ARRAY2,N2)

PURPOSE Moves bytes (or characters) from one array to another.

INPUT

CNT	Byte count.
ARRAY1	Array containing bytes to be moved.
N1	First byte position in ARRAY1 to be moved (N1≥ 0).
N2	First byte position in ARRAY2 to receive data (N2≥ 0).

OUTPUT ARRAY2 Array receiving bytes.

COMMENTS Byte indices start at 0.

NXTCHAR

SYNTAX

NXTCHAR(SWT)

PURPOSE

Accesses next character to be scanned, outputs the character type, and sets up scan globals.

INPUT

SWT Switch controls screening of blanks and case conversion:

0 Access next character, converts lower to upper case.
1 Access next non-blank character, converts lower to upper case.
2 Access next non-blank character — no case conversion.
3 Access next character — no case conversion.

OUTPUT

- System overlay global CHAR:

  Next character, in character format. If there are no more characters in the array, zero is output.

- System overlay global CHARTYP:

Output	*Character Type*
1	Alphabetic or ampersand (&).
2	Number.
3	Special character, or end of array.

NOTES

- INITCHAR and NXTCHAR set up character string and system globals for scan routines.

- The two routines are automatically called to set up scanning of the command line. They must be explicitly called to initialize scanning of any other stream of characters (i.e., a line input with TYPIN).

NXTFPG

SYNTAX NXTFPG(ARRAY,IND,SIZE,OPTIONS)

PURPOSE Accesses the next page of a file that has been initialized with PAGFIL.

INPUT
- ARRAY The eight-word array used for paging this file.
- OPTIONS Same as in FREAD.

OUTPUT
- IND
  - 0 This is not last page.
  - −1 This is last page.
  - 1 Last page already returned.
- SIZE Number of words on the page.

NXTNAM

SYNTAX NXTNAM(NMLST,NINFO,NMBYT,BYTCT)

PURPOSE In a scan initialized with INITNM, NXTNAM gets the next name from the name list.

INPUT
- NMLST Same form used in INITNM.
- NINFO: Same form used in INITNM.

OUTPUT
- NMBYT Byte number in NMLST(2) where name starts.
- BYTCT Byte count of the name.

FUNCTION RETURN
- 0 Not the last name in the list.
- 1 The last name in the list.
- −1 There are no more names in the list.

COMMENT NXTNAM is part of a package that includes FMNLST and INITNM. Call FMNLST to get the list of names, and then call INITNM and NXTNAM to get individual names from the list.

OCTBCD

SYNTAX OCTBCD(OCT,TXT)

PURPOSE Converts an integer to an octal text string.

INPUT OCT Integer for conversion.

OUTPUT TXT Three-word array receiving the octal text string (six characters are always returned).

OREAD

SYNTAX OREAD(CHANNEL,DLOC,COUNT,BUF,OPTIONS)

PURPOSE Reads an overlay into memory. If there is already an overlay in memory, it will overwrite.

NOTES

- All arguments are the same as FREAD, see FREAD for further documentation.
- Use OREAD with the Trace or Debug options of the loader.

PAGFIL

SYNTAX PAGFIL (CHANNO,DLOC,SECTORS,BUFFER,PAGSIZ,ARRAY)

PURPOSE Initialize file paging so that a file opened for read may be paged.

INPUT

CHANNO	The channel number of the open file.
DLOC	Starting disc location (two WORDS, as in FREAD).
SECTORS	Number of sectors to be paged (two words).
BUFFER	Buffer into which file is to be paged.
PAGSIZ	Size of the pages in sectors ($\leq$ BUFFER size in sectors).
ARRAY	Eight-word array used to page file.

PERROR

SYNTAX PERROR(ERROR,FILENAME,LEVEL)

PURPOSE Prints an error message, given an error number and a text file of messages.

INPUT

ERROR Error number.

FILENAME Filename in FMNAME format or zero if there is no file.

LEVEL Error level:

- −1 User selectable default. See the SELECT ERRLEVEL command in the *CGOS 200 GNA Operator Guide*.
- 0 Print the error number only.
- 1 Print the error number and the short message.
- 2 Print the error number and the long message.
- 3 Print the short message only.
- 4 Print the long message only.

NOTES

- Error messages are referenced in text files. The first file searched is supplied by the caller (argument FILENAME). If no message is found there, the system files are checked. (The system files are: SYSNEWS.ERROR/FM,SYS.)
- The error message has three parts:

  1. A hex number (printed when LEVEL is 0, 1, or 2). The format is:

     ERROR CALL xxxx

  2. A short message, printed if LEVEL is not 0.

  3. Text that is printed if LEVEL is 2 or 4.

OUTPUT Message text file as follows:

The first line is six characters long and contains the hex error number. The format is:

==xxxx

OUTPUT (Continued)

The lines that follow constitute the message, which is terminated by a line starting with "= =".

There are two varieties of message lines. Lines that begin with "--" are part of the short message. All other lines are part of the long message.

COMMENTS

- Short and long message lines may occur in any order.
- For an example of a message text file, see SYSNEWS.ERROR.SYS.

PNCHLDR

SYNTAX	PNCHLDR(CNT)
PURPOSE	Punches a leader on paper tape unit.
INPUT	CNT Number of blanks (no holes).
NOTE	Each blank occupies one frame = 1/10th inch.

POP

SYNTAX	POP(P,WORD)
PURPOSE	Removes a word from the top of a push-down list.
INPUT	P Push-down list.
OUTPUT	WORD Word removed.
FUNCTION RETURN	Word removed.
NOTE	See IPDL, PUSH.

POPN

SYNTAX	POPN(P,CNT,ARR)	
PURPOSE	Removes a block of words from a push-down list.	
INPUT	P	Push-down list.
	CNT	Number words to remove.
OUTPUT	ARR	Words removed from push-down list.
NOTE	See IPDL, PUSHN.	

PPT

SYNTAX	PPT(CNT,ARR)	
PURPOSE	Outputs data to punch paper tape unit (PPTDEV).	
INPUT	CNT	Word Count.
	ARR	Array of words output to PPTDEV.
ERRORS	'F00E'x	No PPTDEV assigned.
NOTE	The left half of a word is punched first.	

PPT1

SYNTAX PPT1(CHAR)

PURPOSE Outputs one character to PPTDEV, if PPTDEV is assigned.

INPUT CHAR Character to be output, in byte format.

ERRORS 'F00E'x = No PPTDEV assigned.

PPTN

SYNTAX PPTN(CNT,ARR)

PURPOSE Outputs a string of bytes to PPTDEV.

INPUT CNT Number of bytes to be output.

ARR Byte string packed two bytes per word — starting in left byte of first word.

ERRORS 'F00E'x = No PPTDEV assigned.

PUSH

SYNTAX PUSH(P,WORD)

PURPOSE Adds a single precision integer to the top of a push-down list.

INPUT
- P Push-down list
- WORD Word to add.

COMMENTS See IPDL, POP.

PUSHN

SYNTAX PUSHN(P,CNT,ARR)

PURPOSE Adds a block of words to a push-down list.

INPUT
- P Push-down list.
- CNT Number of words to add.
- ARR Words to add.

COMMENTS See IPDL, POPN.

PUTBIT

SYNTAX PUTBIT(VALUE,STRING,INDEX)

PURPOSE Turns a bit ON or OFF.

INPUT
- VALUE 0 Bit OFF, otherwise bit is ON.
- STRING Bit string array.
- INDEX Bit index into STRING (leftmost bit is 0).

THE PUTFILE UTILITY PACKAGE

The PUTFILE utility package is used to access text files on a line-by-line basis. The normal sequence for reading from a text file would be open with PUTFILE, read lines of text with PUTLINE or PUTLINEB, and close with PUTCLOSE, PUTCLEAR, and PUTABORT.

PUTFILE

SYNTAX

PUTFILE(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE

Opens a text file for output on a line-by-line basis.

INPUT

FSB — File Status Block (eight words, see below) FSB(1) MUST be set to 1.

BUF — Output Buffer.

BUFSIZ — Length of buffer in sectors.

FNAME — Filename (without the &BCD) in the form returned by FNAME:

Word 1	Character count (includes !).
Word 2-n	Filename (includes !).

OPTIONS Bit-oriented options:

Bit		(Leftmost bit is 0.)
7	ON	Create new catalogs as required.
	OFF	Give error if any catalog missing.
15	ON	Function returns errors.
	OFF	Errors go to System error routine.

Description of FSB (File Status Block):

FSB(1)	Status:	1 No file open.
		12 File open.
		13 File open without BUF in memory.
FSB(2)	Output channel number.	
FSB(3-4)	Relative DLOC in file.	
FSB(5)	Byte counter.	
FSB(6)	Old line count.	
FSB(7)	Number of words in buffer.	
FSB(8)	Running CHECKSUM for the output line.	

PUTFILE

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

NOTE

None of the PUTFILE routines have local variables to maintain. Only the integrity of FSB and BUF is important.

RESTRICTIONS

- BUFSIZ is a multiple of SECTSIZ that cannot exceed 32K bytes (16384 words).
- BUFSIZ must remain constant during the writing of the text file. However, these routines can be used to write two files at the same time, using a BUFSIZ of two sectors for one file and five sectors for the other.

PUTLINE

SYNTAX

PUTLINE(FSB,BUF,LEN,LINE)

PURPOSE

Adds a line to the text file opened by PUTFILE.

INPUT

FSB	File Status Block.
BUF	Disc buffer.
LEN	Length of the character string in LINE.
LINE	Character string.

NOTES

- Additional documentation, including a description of the FSB (File Status Block), may be found under PUTFILE.
- This routine must operate in conjunction with PUTFILE.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

PUTLINEB

SYNTAX PUTLINEB(FSB,BUF,LEN,LINE,OFFSET)

PURPOSE Adds a line to the text file opened by PUTFILE. Equivalent to PUTLINE with an added byte offset.

INPUT

FSB	File Status Block.
BUF	Disc buffer.
LEN	Length of the character string in LINE.
LINE	Character string.
OFFSET	Byte offset in LINE (counted from 0) where the string of LEN bytes will be written.

NOTES

- Additional documentation, including a description of the FSB (File Status Block), may be found under PUTFILE.
- This routine must operate in conjunction with PUTFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

PUTABORT

SYNTAX — PUTABORT(FSB,BUF)

PURPOSE — Aborts the text file being written by PUTFILE.

INPUT — FSB File Status Block.

BUF Disc buffer.

NOTES
- Complete documentation, including a full description of the FSB (File Status Block), may be found under PUTFILE.
- This routine must operate in conjunction with PUTFILE.

ERRORS — For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

PUTCLEAR

SYNTAX — PUTCLEAR(FSB,BUF)

PURPOSE — Marks the FSB (File Status Block) of an open file to indicate that the text file buffer is no longer available and must be loaded into memory the next time PUTLINE is called.

INPUT — FSB File Status Block.

BUF Disc buffer.

NOTES
- Additional documentation, including a description of the FSB (File Status Block), may be found under PUTFILE.
- This routine must operate in conjunction with PUTFILE.

ERRORS — For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

PUTCLOSE

SYNTAX PUTCLOSE(FSB,BUF)

PURPOSE Closes the text file being written by PUTFILE.

INPUT FSB File Status Block (see PUTFILE).

BUF Disc buffer.

NOTES
- Additional documentation, including a description of the FSB (File Status Block), may be found under PUTFILE.
- This routine must operate in conjunction with PUTFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

PUTFLD

SYNTAX PUTFLD(VALUE,STRING,FBIT,SIZE)

PURPOSE Puts a specific value into one field of a bit string.

INPUT

VALUE — Integer value to be stored. Only the low order (rightmost) bits are used.

STRING — Bit string to store VALUE.

FBIT — Location of field where VALUE will be stored. Location is specified by the displacement between the high-order (left most) bit of the field and the high-order bit of string.

SIZE — Size of target field in bits (0 < SIZE < 17).

OUTPUT Side effects of changing STRING or any error call due to inconsistent arguments.

The change to STRING can be characterized in two ways:

- GTFLD.(STRING,FBIT,SIZE) will return VALUE.
- For I=1,SIZE, STRING (FBIT+I−1) will equal value (15-SIZE+I) when STRING and VALUE are assumed to be bit vectors.

PUTLABEL

SYNTAX PUTLABEL(TBLABEL,SAVLABEL)

PURPOSE Puts the label saved in SAVLABEL back into the requested system label spot in the TASKBLOCK.

INPUT

TBLABEL Identifies the desired system label to put into the TASKBLOCK:

-1	TAPERLBL
-2	SYSERLBL
-3	INTRQUIT

SAVLABEL Two-word array for the label output by GETLABEL, or a local label created by calling LBLSET.

ERROR 'F00D'X — TBLABEL specified is illegal (bad arg).

NOTES

- For safety, PUTLABEL should only be used in the user address space.
- See also, GETLABEL, LBLGO, LBLSET, and RESETLBL.

THE READBFIL UTILITY PACKAGE

The general sequence for reading bytes from a binary sequential file would be to open with READBFIL, read bytes with READBYTE, and close with READCLS.

READBFIL

SYNTAX

READBFIL(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE

Opens binary sequential file for input.

INPUT

FSB(11) File Status Block;
FSB(1) must be set to one.

BUF(255) Disc buffer.

BUFSIZ Size of disc buffer in sectors.

FNAME Filename in the form returned by FMNAME:

Word 1	Character count (includes !).
Word 2-n	Filename (includes !).

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended file information array in OPTIONS(2-16).
14	ON	Return normal file information array in OPTIONS(2-5).
15	ON	Error condition results in error FLAG returned as a function name.
	OFF	Error condition results in the call ERROR(FLAG).

OUTPUT (Continued)

OPTIONS(1) bit 14 OFF:

(Word)	Normal Array
(2-3)	Disc location, always (0,0).
(4-5)	Total sector count of file,

OPTIONS(1) bit 14 ON:

(Word)		Extended Array.
(2-3)		Disc location, always (0,0).
(4-5)		Total sector count of file.
(6)		Creation date.
(7)		Time of day of creation in one-second intervals.
(8)		Access date.
(9)		File protection group:
	Bit	(Leftmost bit is 0.)
	0-7	Reserved.
	8-15	File protection group.
(10)		File status, with bit ON:
	Bit	(Leftmost bit is 0.)
	0	In use.
	1	Opened for modify in place.
	2	To be deleted when closed if use count is 0 (i.e. old file of create with supersede).

OUTPUT (Continued)		3	Opened for create.
		4	Temporary entry for create with supersede.
		5	Opened for regular read.
		6	Reserved.
		7	File is a catalog.
		8	File is a volume.
		9-15	Reserved.
	(11)		Number of users currently accessing file or included file — incremented each time the file or included file is opened, decremented each time it is closed.
	(12)		Contiguous data flag:
		O	All data allocated contiguously.
		N	Data not contiguous, N is sector count (N >0) of first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).
	(13)		User-defined attribute value.
	(14)		User-defined attribute value.
	(15)		CHKSUM.
	(16)		File type.
		Bit	(Leftmost bit is 0.)
		0-7 0	Not task protected.
		>0	1 + number of task which has protected the file.
		8-15 0	Not defined.
		1	Catalog.
		2	Object.
		3	Text.
		4	Configuration.
		5	Reserved.
		6	PEP object code.
		7	Core image (overlay).
		8	Command table.
		9	Loader data file (symbol table or library).

OUTPUT (Continued)

A	Reserved.
B	Accounting table.
E	Reserved.
F	Z80 binary files.
10-1F	Reserved for CGOS.
20	CADDS 4 part files.
21	CADDS 4 TVF files.
22	CADDS 4 figure files.
23-2F	Reserved for CADDS 4.
30-AF	Unused.
B0-BF	Reserved for batch files.
C0-FE	Unused.
FF	Work files (deleted on FMCLEAR).

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

Refer to ERROR. For description of FILE MANAGER ERRORS.

READBYTE

SYNTAX	READBYTE(FSB,BUF,COUNT,BLOCK,OFFSET,ACTUAL)	
PURPOSE	Retrieves the next block of bytes from a binary sequential file.	
INPUT	FSB	File Status Block.
	BUF	Disc buffer.
	COUNT	Number of bytes to read.
	OFFSET	Byte offset in output block.
	ACTUAL	Actual number of bytes transferred.
OUTPUT	BLOCK	Block of bytes.
NOTE	This routine must operate in conjunction with READBFIL.	

READBCLS

SYNTAX	READBCLS(FSB,BUF)	
PURPOSE	Closes the input channel of a binary sequential file.	
INPUT	FSB	File Status Block.
	BUF	Disc buffer.
NOTE	This routine must operate in conjunction with FM.READBFIL, Additional documentation can be found under READBFIL.	

THE READFILE UTILITY PACKAGE

The general sequence for reading a binary sequential file would be to open with READFILE, read blocks of words with READBLOK or READSECT, and close with READCLOS.

These four routines can also be used to read a random access file. There are three additional functions for manipulating the sequential access pointer of a random access file. A call to READMARK reads the position of the pointer, while READPOS and READTOP can be used to move the pointer.

READFILE

SYNTAX

READFILE(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE

Opens a binary sequential file for reading.

INPUT

FSB(11) File Status Block, FSB(1) must be set to one before calling READFILE. After a call to READCLOS, the value of FSB(1) will also be one. In the interim, FSB may not be read or changed.

BUF(255) Disc buffer.

BUFSIZ Size of disc buffer in sectors.

FNAME Filename in FM name format:

Word 1	Character count (includes !).
Word 2-n	Filename (includes !).

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended file information array in OPTIONS(2-16).
14	ON	Return normal file information array in OPTIONS(2-5).
15	ON	Error condition results in error FLAG returned as a function name.
OFF		Error condition results in the call ERROR(FLAG).

OUTPUT

OPTIONS(1) bit 14 ON:

(Word)		Normal Array.
(2-3)		Disc location, always (0,0).
(4-5)		Total sector count of file.

OPTIONS(1) bit 14 OFF:

(Word)		Extended Array.
(2-3)		Disc location, always (0,0).
(4-5)		Total sector count of file.
(6)		Creation date.
(7)		Time of day of creation in one-second intervals.
(8)		Access date.
(9)		File protection group.
	Bit	(Leftmost bit is 0.)
	0-7	Reserved.
	8-15	File protection group.
(10)		File status, with bit ON:
	Bit	(Leftmost bit is 0.)
	0	In use.
	1	Opened for modify in place.
	2	To be deleted when closed if use count is 0 (i.e., old file of create with supersede).
	3	Opened for create.
	4	Temporary entry for create with supersede.
	5	Opened for regular read.
	6	Reserved.
	7	File is a catalog.
	8	File is a volume.
	9-15	Reserved.

OUTPUT (Continued)

(11) Number of users currently accessing file or included file — incremented each time the file or included file is opened, decremented each time it is closed.

(12) Contiguous data flag:

0	All data allocated contiguously.
N	Data not contiguous, N is sector count (N 0) of first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).

(13) User-defined attribute value.

(14) User-defined attribute value.

(15) CHKSUM.

(16) File type

Bit		(Leftmost bit is 0.)
0-7	=0	Not task protected.
	>0	1 + number of task which has protected the file.
8-15	0	Not defined.
	1	Catalog.
	2	Object.
	3	Text.
	4	Configuration.
	5	Reserved.
	6	PEP object code.
	7	Core image (overlay).
	8	Command table.
	9	Loader data file (symbol table or library).
	A	Reserved.
	B	Accounting table.
	C-E	Reserved.
	F	Z80 binary files.
	10-1F	Reserved for CGOS.
	20	CADDS 4 part files.
	21	CADDS 4 TVF files.
	22	CADDS 4 figure files.

OUTPUT (Continued)

23-2F	Reserved for CADDS 4.
30-AF	Unused.
B0-BF	Reserved for batch files.
C0-FE	Unused.
FF	Work files (deleted on FMCLEAR).

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

Refer to ERROR for description of FILE MANAGER ERRORS.

READBLOK

SYNTAX READBLOK(FSB,BUF,COUNT,BLOCK)

PURPOSE Retrieves the next block of words from a binary sequential file.

INPUT
- FSB File Status Block.
- BUF Disc buffer.
- COUNT Number of words to read.

OUTPUT
- BLOCK Block of words.

COMMENT This routine should operate in conjunction with READFILE. Complete documentation can be found there.

READSECT

SYNTAX READSECT(FSB,DLOC,COUNT,BLOCK)

PURPOSE Reads a block of words from a file opened for sequential word access. The block begins on a sector boundary specified by the user.

INPUT
- FSB File Status Block.
- DLOC Disc location to begin reading from.
- COUNT Number of words to read.

OUTPUT
- BLOCK Block of words.

NOTE This routine must be used in conjunction with READFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

READCLOS

SYNTAX READCLOS(FSB,BUF)

PURPOSE Closes the channel of a binary sequential file.

INPUT FSB File Status Block.

BUF Disc buffer.

NOTE This routine must operate in conjunction with READFILE.

READMARK

SYNTAX READMARK(FSB,BUF,WORD)

PURPOSE Returns the current position in a sequential binary file.

INPUT FSB File Status Block.

BUF Disc buffer.

OUTPUT WORD Double integer indicating position of next word in file.

NOTE This routine should be used in conjunction with READFILE.

READPOS

SYNTAX READPOS(FSB,BUF,DLOC,INDEX)

PURPOSE Sets the sequential access pointer to a new disc location and sector index.

INPUT
- FSB File Status Block.
- BUF Disc buffer.
- DLOC Disc location to read from.
- INDEX Sector index to read from.

NOTE This routine must be used in conjunction with READFILE.

ERRORS For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM

READTOP

SYNTAX READTOP(FSB,BUF)

PURPOSE Moves sequential access pointer to beginning of file.

INPUT
- FSB File Status Block (see READFILE).
- BUF File buffer (see READFILE).

COMMENT READTOP is part of the READFILE utility and must be used in conjunction with READFILE.

RENAME

SYNTAX RENAME(OLDNAME,NEWNAME,OPTIONS,SPECFIR)

PURPOSE

- Renames lowest level catalog or filename (delete/write access).
- Changes file protection group (change access). Changes user attribute (write access).
- Returns a value to indicate that file exists; called through FILEXS — exclusive operation (read access).
- Assigns or deassigns task protection — no other bits (read access) can be set.
- Returns SRCHFIL or file entry ARRAY (read access), file must not be in use on any OPTION except unassigning task protection.

INPUT

OLDNAME — Full ASCII filename with a (.) as the delimiter between components and an (!) following the last character. Each component of the filename must be 20 characters or less.

NEWNAME — New ASCII filename, to replace the rightmost component of the OLDNAME. The name must be 20 characters or less, excluding the (!), which follows the last character and terminates the string.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Change rightmost component to NEWNAME.
1	ON	Change the file protection group to value in SPECIFR(1).
2-3		Reserved (must be zero).
4	ON	Change user attribute value to SPECIFR(4-5).
5	ON	F,N first word of SRCHFIL ARRAY (ignore other bits in OPTIONS(1) except 15).

INPUT
(Continued)

Note

Options for bits 6 and 7 are reserved for system use only.

6	ON	Return SRCHFIL ARRAY (has precedence over bit 7) also F,N as bit 5 OPTIONS(1).
7	ON	Return file entry ARRAY (if bit 6 is OFF) also F,N as bit 5, OPTIONS(1).
8	ON	Return total file size (= -1 if entry is a volume) also F,N as bit 5, OPTIONS(1).
9		Reserved (must be zero).
10	ON	Assign task protection. No effect if: a. Any of bits 0-9,11-14 are also on. b. File is already task protected. c. File is in use (gives error C017). d. File is access protected (gives error C012).
11	ON	Unassign task protection. No effect if: a. Any of bits 0-10,12-14 also ON. b. File not task protected. c. If task protected but not by this task. d. File is access protected (give error C012).
12		Reserved (must be zero).
13	ON	OPTIONS(2) contains additional options data.
	OFF	OPTIONS(2) ignored.
14	ON	Return working directory name count in OPTIONS(3) followed by characters named in OPTIONS(4-42). If working directory not enabled, value is returned in OPTIONS(3).
15	ON	Error condition results in error FLAG returned as function name.
	OFF	Error condition results in the call ERROR(FLAG).

INPUT (Continued)

OPTIONS(2)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero).
SPECIFR(1)	New file protection group.
(4-5)	New value of user attribute.

OUTPUT

OPTIONS(1) bit 14 ON:

OPTIONS(3)	Directory name character count or < 0 if working directory not enabled.
OPTIONS(4-42)	Directory name for OPTIONS(3) number of characters

OPTIONS(1) bit 6 ON:

SPECIFR(Word)		SRCHFIL ARRAY
(1)	= −3	Entry is a volume.
	= −2	Entry is a catalog.
	= −1	Entry found — not a catalog.
	= 0	File not found in parent catalog.
	> 0	Catalog in complete filename does not exist. Value is level number of first catalog that did not exist (1 = leftmost catalog in filename string).
(2-3)		DLOC of file entry sector.
(4)		Index into file entry sector.
(5-6)		DLOC of parent catalog sector.
(7)		Index into parent catalog sector.

OUTPUT (Continued)

OPTIONS(1) bit 7 ON:

SPECIFR(1-25)	File entry array.
SPECIFR(26-27)	Volume DLOC (includes unit and drive).

OPTIONS(1) bit 8 ON:

SPECIFR(1-2)	Total file size or −1 for volume.

FUNCTION RETURN

If OPTIONS(1) bit 15 is set:

0	Indicates no error.
FLAG	Indicates type of FILE MANAGER ERROR.

ERRORS

For a description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

RESETLBL

SYNTAX — RESETLBL(TBLABEL)

PURPOSE — Resets requested system label to the task default value specified in SYSOV.NEXTCOMM. The label is assigned this value when a new command is issued; the value may be changed with LBLSET or PUTLABEL.

INPUT — TBLABEL Label to be reset:

−1	TAPERLBL
−2	SYSERLBL
−3	INTRQUIT

ERRORS — 'F00D'x TBLABEL specified is illegal (bad arg).

COMMENTS

- For safety, RESETLBL should only be used in the user address space.
- See also, GETLABEL, PUTLABEL, LBLGO, and LBLSET.

RETERR

SYNTAX — RETERR(ERRNUM)

PURPOSE — Returns latest system error number after a call to ERROR has forced a LBLGO(SYSERLBL).

OUTPUT — ERRNUM System error number.

FUNCTION RETURN — System error number.

ROPEN

SYNTAX ROPEN(CHANNEL,FILENAME,OPTIONS,ARRAY)

PURPOSE Opens a file to be read.

Note

Read access is necessary. File cannot be open for modify.

INPUT FILENAME

(Word)

(1-41)	Full ASCII filename with a period (.) between components and an exclamation mark (!) following the last character. Each component of the filename must be 20 characters or less.
(42-43)	DLOC of file entry sector.
(44)	Index of file entry.

OPTIONS(1)

Bit		(Leftmost bit is 0.)
0	ON	Return extended file information array.
	OFF	Return normal file information array.
1	ON	No-search option enabled. FILENAME (41-43) contains DLOC and index including volume info for file entry.

Note

This option is reserved for system use only.

	OFF	No-search-option disabled.
2-12		Reserved (must be zero).

INPUT (Continued)

2-12		Reserved (must be zero).
13	ON	OPTIONS(2) contains additional option data.
	OFF	OPTIONS(2) ignored.
14	ON	Return working directory name count in OPTIONS(3), followed by name characters in OPTIONS(4-42). If working directory disabled, value is returned in OPTIONS(3).
15	OFF	Error condition results in the call ERROR(FLAG).

OPTIONS(2)

Bit	(Leftmost bit is 0.)
0-15	Reserved (must be zero).

OUTPUT

OPTIONS(1) bit 14 ON:

OPTIONS(3)	Directory name character count or ≤0 if working directory not enabled.
OPTIONS(4-42)	Directory name for OPTIONS(3) number of characters.
ARRAY (Word)	File information array.
OPTIONS(1)	Bit 0 OFF — Normal Array.
(1-2)	Disc location, always (0,0).
(3-4)	Total sector count of file.
OPTIONS(1)	Bit 0 ON — Extended Array.
(1-2)	Disc location, always (0,0).
(3-4)	Total sector count of file.
(5)	Creation date.
(6)	Time of day of creation in one-second intervals.
(7)	Access date.

OUTPUT (Continued)

(8) File protection group.

Bit	(Leftmost bit is 0.)
0-7	Reserved.
8-15	File protection group.

(9) File status, with bit ON:

Bit	(Leftmost bit is 0.)
0	In use.
1	Opened for modify in place.
2	To be deleted when closed if use count is 0 (i.e., old file of create with supersede).
3	Opened for create.
4	Temporary entry for create with supersede.
5	Opened for regular read.
6	Reserved.
7	File is a catalog.
8	File is a volume.
9-15	Reserved.

(10) Number of users currently accessing file or included file — incremented each time the file or included file is opened, decremented each time it is closed.

(11) Contiguous data flag:

0	All data allocated contiguously.
N	Data not contiguous, N is sector count (N > 0) of first contiguous chunk of the file (if file has 0 length this is indicated by ARRAY(3-4)).

(12) User-defined attribute value.

(13) User-defined attribute value.

(14) CHKSUM

OUTPUT (Continued)

(15) File type:

Bit	(Leftmost bit is 0.)	
0-7	=0	Not task protected.
	< 0	1 + number of task which has protected the file.
8-15	0	Not defined.
	1	Catalog.
	2	Object.
	3	Text.
	4	Configuration.
	5	Reserved.
	6	PEP object code.
	7	Core image (overlay).
	8	Command table.
	9	Loader data file (symbol table or library).
	A	Reserved.
	B	Accounting table.
	C-E	Reserved.
	F	Z80 binary files.
	10-1F	Reserved for CGOS.
	20	CADDS 4 part files.
	21	CADDS 4 TVF files.
	22	CADDS 4 figure files.
	23-2F	Reserved for CADDS 4.
	30-AF	Unused.
	B0-BF	Reserved for batch files.
	C0-FE	Unused.
	FF	Work files (deleted on FMCLEAR).

CHANNEL Channel number assigned to the file.

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0 Indicates no error.

FLAG Indicates type of FILE MANAGER ERROR.

ERRORS

Refer to ERROR for description of FILE MANAGER ERRORS.

RPT

SYNTAX RPT(SWT,CNT,ARR)

PURPOSE Inputs data from read paper tape unit (RPTDEV).

INPUT

SWT	0	Read tape from first (current) character without skipping leader. First character goes in left byte of first word.
	1	Read tape from first non-blank character skip leader. First character goes in right byte of first word; left byte = '00'x.
	2	Read tape from first non-blank character skip leader. First character goes in left byte of first word.
CNT		Word count (number of words to be filled).

OUTPUT ARR Array receiving the data.

ERRORS 'F00E'x No RPTDEV assigned.

RPT1

SYNTAX RPT1(CHAR)

PURPOSE Inputs one character from RPTDEV, if RPTDEV is assigned.

OUTPUT CHAR Character input, in byte format.

ERRORS 'F00E'x = No RPTDEV assigned.

RPTN

SYNTAX	RPTN(CNT, ARR)	
PURPOSE	Inputs bytes from RPTDEV.	
INPUT	CNT	Number of bytes to input.
	ARR	Array containing bytes input, packed one per right byte of word.
ERRORS	'F00E'X =	No RPTDEV assigned.

SELESORT

SYNTAX SELESORT(ARRAY,N,COMPFUN)

PURPOSE Sorts an internal file of one-word records. The comparison of keys is performed by a user-supplied function.

INPUT

ARRAY — Internal file of one-word records.

N — Number of records in file.

COMPFUN — Record comparison function with a return that depends on the relation of KEY1 and KEY2:

KEY1	*COMPFUN*
< KEY2	< 0
= KEY2	= 0
> KEY2	> 0

OUTPUT

ARRAY — Sorted internal file.

NOTE Method:

Knuth, Donald E., *The Art of Computer Programming,* First Edition, Section 5.

Modified for TPL (arrays start at subscript 0):

S1 (LOOP ON J) PERFORM STEPS S2 AND S3 FOR
J = N,N-1,...0.
S2 (FIND MAX (K1,...,KJ) SEARCH THRU KEYS KJ,KJ-1,K1
TO FIND A MAXIMAL ONE; LET IT BE KI.
S3 (EXCHANGE WITH RJ) INTERCHANGE RECORDS RI< = = >RJ.
(NOW RECORDS RJ,...,RN ARE THEIR FINAL POSITION.

SETBIT

SYNTAX SETBIT(STRING,INDEX)

PURPOSE Turns on a specified bit in a bit string.

INPUT STRING Bit string array.

INDEX Bit STRING number (leftmost bit is 0).

SETBREAK

SYNTAX SETBREAK

PURPOSE Instructs CHKINTR, CHKQUIT, and CHKSTOP to set a flag to be tested by CHECKBREAK instead of executing an LBLGO.(INTQUIT).

NOTE See CHKBREAK.

SETCOM

SYNTAX SETCOM(LEN,STRING,IPOS)

PURPOSE Inserts a string in the task command buffer. To view the string, use a task option to STATUS.

INPUT

LEN	Length of the string.
STRING	String to be inserted.
IPOS	Buffer location for inserting STRING.

Note

If STRING is too long for the command buffer, characters will be inserted until the length limitation is exceeded.

OUTPUT String inserted into the command buffer.

SETPG

SYNTAX

SETPG(BUFFER,SPID,PERMIT)

PURPOSE

Modifies a page slot of the user's page frame. The user specifies a new page and the permitted access for the page slot.

INPUT

BUFFER — Memory buffer:

- One memory page long ('800'x words).
- Aligned on a page boundary.

SPID — System Page ID of the page to put into the slot specified by BUFFER. If SPID is NULL, the new slot will be empty.

PERMIT — Type of access for the page slot identified by BUFFER:

P&RW	Any access is legal.
P&RO	Read-only.
P&NO	No legal access.

FUNCTION RETURN

0	No errors.
E010	BUFFER is not aligned on a page boundary.
E020	Illegal system page identifier (SPID).
E050	PERMIT allows reading or writing the page, but SPID specifies the NULL SPID.
E051	The page specified by SPID is protected and cannot be accessed in the manner requested by PERMIT.
E060	Illegal protection attribute (PERMIT).
E110	Cannot modify O/S page slot.

NOTES

- The symbolic constants are defined in SYM.EQU.MEMMAN and SYM.EQUF.MEMMAN.
- The caller must have logical access to the memory page specified by SPID. If the caller owns the page, the owner protection attributes must allow the access requested by PERMIT. Otherwise, the public protection attributes must allow access requested by PERMIT. O/S page slots may not be modified.

SETPGP

SYNTAX SETPGP(SPID,PROT)

PURPOSE Modifies the parameters of a system page.

INPUT

SPID	System Page ID of a memory page.	
PROT	Desired protection attributes:	
	P&RW	Any access is legal.
	P&RO	Read-only.
	P&NO	No legal access.

FUNCTION RETURN

0	No errors.
E020	Illegal system page identifier (SPID).
E024	Calling task does not own the page specified by SPID.
E060	Illegal protection attribute (PROT).

NOTE The symbolic constants are defined in SYM.EQU.MEMMAN and SYM.EQUF.MEMMAN.

SETPUNCH

SYNTAX SETPUNCH(VALUE)

PURPOSE Sets the parity option for the task PPTDEV.

INPUT VALUE Possible parity OPTIONS:

0	No parity, transparent data transfer.
1	Odd parity.
2	Even parity.
3	Mark — MSB of byte always set to 1.
4	Space — MSB of byte always set to 0.

FUNCTION RETURN

0	No errors.
1	Not used.
2	Unit number is not valid.
3	Unit cannot be a PPTDEV.
4	Value is not in range 0-4.
5	Unit not declared PPTDEV.

SETSTRG

SYNTAX SETSTRG(STRING)

PURPOSE Copies eight characters into the task global TSKUSTRG.

INPUT STRING Eight-byte string to be stored in TSKUSTRG.

OUTPUT STRING is moved into the task global TSKUSTRG.

SHELSRTN

SYNTAX — SHELSRTN(RECS,NREC,COMPFUN,RECLEN,REC)

PURPOSE — Sorts records with a shell sort. Timing tests show that SHELSRTN is the fastest general sort for an array of 15 to 1500 multiword records. It is not stable, however. For a stable sort, use INSESRTN.

INPUT

RECS — Array containing data to be sorted.

NREC — Number of records.

COMPFUN — Record comparison function with a return that depends on the relation of KEY1 and KEY2:

KEY1	*COMPFUN*
< KEY2	< 0
= KEY2	= 0
> KEY2	> 0

RECLEN — Length of record (in words).

REC — Caller supplied scratch, RECLEN words long.

OUTPUT — Array RECS is sorted.

NOTE — The sorting method is described in *Knuth, Vol.3, pg. 85.* SHELSRTN uses the increments suggested at the bottom of p.95. Comments of the form 'DN' reflect the algorithmic steps outlined on p.85.

SNGL

SYNTAX — SNGL(F)

PURPOSE — Converts double precision real to real.

INPUT — F Double precision real.

FUNCTION RETURN — Single precision real.

SRTESTIO

SYNTAX — SRTESTIO(IOFLAG,ERRNO)

PURPOSE — Tests an IOFLAG for completion of no-wait I/O.

INPUT — IOFLAG Flag returned by an I/O request call.

OUTPUT — ERRNO Error value returned by the I/O system.

FUNCTION RETURN

0 I/O not done.
−1 I/OFLAG is bad or flag was not assigned to a request
1 I/O done and flag released, or no IOFLAG's for request

NOTES

- If the FUNCTION RETURN is zero, SRTESTIO should be called again later.
- ERRNO value is not valid with an I/O request in progress (FUNCTION RETURN = 0).

SRWAITIO

SYNTAX SRWAITIO(IOFLAG,ERRNO)

PURPOSE Changes a NOWAIT I/O request into a WAIT I/O request.

INPUT IOFLAG Address of a valid IOFLAG.

OUTPUT ERRNO Error value generated by the I/O system.

NOTE SRWAITIO resets the IOFLAG value. This eliminates the need for calling TESTIO.(IOFLAG).

STBYTF

SYNTAX STBYTF(WORD,ARRAY,N)

PURPOSE Stores a byte (FORTRAN version).

INPUT WORD Word containing byte, in byte format.

N Byte position in the array (N≥ 1).

OUTPUT ARRAY Array receiving the byte.

NOTE Byte indices start at 1.

STBYTT

SYNTAX STBYTT(WORD,ARRAY,N)

PURPOSE Stores a byte (TPL version).

INPUT
- WORD — Word containing byte, in byte format.
- N — Byte position in the array (N≥ 0).

OUTPUT
- ARRAY — Array receiving the byte.

NOTE Byte indices start at 0.

STCHRF

SYNTAX STCHRF(WORD,ARRAY,N)

PURPOSE Stores a character (FORTRAN version).

INPUT
- WORD — Word containing character, in character format.
- N — Byte position in the array (N≥ 1).

OUTPUT
- ARRAY — Array receiving WORD.

NOTE Byte indices start at 1.

STCHRT

SYNTAX STCHRT(WORD,ARRAY,N)

INPUT
- WORD — Word containing character in character format.
- N — Byte position in the array (N≥ 1).

OUTPUT
- ARRAY — Array receiving WORD.

NOTE Byte indices start at 0.

STYPEOK

SYNTAX — STYPEOK(FILENAME,MESSG1,MESSG2)

PURPOSE — Checks for OK response to a message printed on the COMDEV. The response must also come from the COMDEV.

Note

If the response is QUIT, then STYPEOK does a LBLGO to −3 −. This is identical to an INTR-QUIT <ESC>Q.

INPUT

FILENAME	Filename in standard FMNAME format, or a third message.
MESSG1	First message: (1) Character count. (2-n) Message characters.
MESSG2	Question asked: TYPE OK TO MESSG2: (1) Character count. (2-n) Message characters.

FUNCTION RETURN

0	Response was OK.
−1	No OK response.

NOTE — If a third message is used and the final character in the message is an exclamation mark (!), the (!) will not be printed (as in a filename).

STYPIN

SYNTAX — STYPIN(MAX,BUF,CNT,LCFLAG,ECHO,EXFLAG)

PURPOSE — Inputs a sequence of characters from the COMDEV or an execute file.

INPUT

MAX — Maximum number of characters.

LCFLAG — Lower case flag:

0	Lower case is accepted.
1	Lower case converted to upper case.
−1	Use default (set by SELECT command).

ECHO — Echoing flag:

0	Echo all characters normally.
1	Echo all characters as spaces.
−1	Use default.

EXFLAG — Execute file flag:

0	Take input from execute files or COMDEV.
1	Take input from COMDEV only.
−1	Use default.

OUTPUT

BUF — Array receiving line of characters.

CNT — Actual character count (CNT ≤ MAX).

FUNCTION RETURN — The terminating character of the line:

0	MAX characters output.
'8D'X	Carriage return.
'93'X	Control-S.

NOTE — See TYPIN.

SUBTRACE

SYNTAX	SUBTRACE(ARG)		
PURPOSE	Toggles trace map output.		
INPUT	ARG	'ON'	Enables trace printout.
		'OF'	Disables trace printout.
FUNCTION RETURN	0		Successful completion.
	−1		Illegal argument passed.
NOTE	See Section 3, Subroutine Trace, for further explanation.		

TAPE

SYNTAX TAPE(UNIT,KYWD)

PURPOSE Processes various tape control requests.

INPUT

UNIT Two character tape unit number for processing request

KYWD Keyword specifying type of request:

Tape Status Checks
(See FUNCTION RETURN below):

TR	Is tape unit ready?
TW	Write-enabled?
TF	Last record?
TL	Is tape at load point?
TP	Type of parity setting.
TB	Packed records?
TG	Checks for EOT ERR.
TT	Checks for 9-track unit.
TE	Is end-of-tape on?
IR	Is tape rewinding?

Tape Control Operations
(No FUNCTION RETURN):

BR	Skip back one record.
FR	Skip forward one record.
BF	Skip back one file.
FF	Skip forward one file.
ER	Erase 2.5 inches of tape.
WF	Write EOF.
RW	Rewind.

Set Tape Control Features
(No FUNCTION RETURN):

SO	Odd parity.
SE	Even parity.
SU	Unpacked records.
SP	Packed records.
SG	Generate an error if EOT passed.
SD	No error if EOT passed.

FUNCTION RETURN

Tape Status Checks only:

Keyword	*Return*	*Meaning*
TR	0	Unit not ready.
TW	0	Write ring in.
TF	0	Last record read was not an end-of-file mark.
TL	0	Not at load point.
TP	0	Odd parity.
TB	0	Unpacked records.
TG	0	End-of-tape error set.
TT	0	Nine-Track unit.
TE	0	End-of-tape not on.
IR	0	Unit not rewinding.

TAPENW

SYNTAX — TAPENW(UNIT,ARGLST)

PURPOSE — Processes a tape request using no-wait I/O.

INPUT

UNIT — Two character tape unit number for processing request.

ARGLST — Two word array:

ARGLST(1)

Type of request. Can have any value of the KYWD argument to the TAPE routine.

ARGLST(2)

Status return from the requested test (in ARGLST (1)) — not valid until TESTIO returns an I/O complete. The value is equivalent to the function return from subroutine TAPE (for KYWD = ARGLST (1)).

FUNCTION RETURN — Address of IOFLAG assigned.

NOTE — This is the no-wait entry. There is no guarantee that the request will actually be issued no-wait, but the I/O flag returned must be tested before the operation or status return can be considered to be valid.

TESTIO

SYNTAX TESTIO(IOFLAG)

PURPOSE Tests an IOFLAG for I/O completion.

INPUT IOFLAG returned by an I/O request call.

FUNCTION RETURN

0	I/O not done.
−1	IOFLAG is bad or no flag assigned to request.
1	I/O done and flag released or no IOFLAG for request.

NOTES

- If the FUNCTION RETURN is zero, TESTIO must be called again later for this IOFLAG.
- Any errors detected during the I/O request will invoke the system error handler. The user must call SRTESTIO to return the error value.

TESTTAPE

SYNTAX TESTTAPE(NAME)

PURPOSE Validates a task unit name for an attached tape unit.

INPUT NAME Two-character name.

FUNCTION

0	NAME is valid task unit name for tape unit.
−1	NAME is invalid or not a tape unit.

TIME

SYNTAX TIME(ARR)

PURPOSE Gives the current time.

OUTPUT

ARR		Current time (four-word array):
	1	Year (right two digits).
	2	Day of year (starting at one).
	3	Minutes of day.
	4	Tenths of seconds.

TOGGLE

SYNTAX TOGGLE(DEVICE,SWT)

PURPOSE Checks real time toggle bits and sets non-real time toggles to their proper values.

INPUT

DEVICE	0	Process real time toggles only.
	1	Also toggle HARDEV according to SWT.
	2	Also toggle COMDEV according to SWT.
SWT	−1	Turn off.
	0	Toggle.
	1	Turn on.

THE TRAVERSE PACKAGE

Traverse Process

TRAVERSE walks through portions of the system file structure. For each file encountered during file structure tree walk, it outputs filename and catalog information.

TRAVERSE visits nodes* of a catalog to processes files. The actual processing, however, is left to the discretion of the calling routine. The calling routine must successively call each of the four Traverse routines (TRAVERSE, NEXTNODE, and ABORTRAV) to perform a tree walk. There are three major processing steps in executing the tree walk.

Step One

First, the catalog must be "activated" with TRAVERSE. This routine initializes the tree walk by setting up the catalog to be traversed. It also establishes the kinds of information returned about nodes and files encountered during the walk. TRAVERSE is called once to start the walk.

Step Two

While traversing the catalog structure, the caller requests the "next" node or file as needed (entry NEXTNODE). NEXTNODE is executed repeatedly until it traverses the entire catalog.

Step Three

When the traverse is completed, "deactivate" the primary catalog using ABORTRAV. ABORTRAV may be called at any time to terminate a Traverse. It must be called to do standard cleanup following an error return.

*In general, a node is equivalent to a catalog. The catalog designated as an argument to the TRAVERSE routine becomes the primary node or tree. All other nodes are sub-trees. A file is a terminating leaf on a tree or sub-tree. If requested, TRAVERSE will return nodes as files.

TRAVERSE

SYNTAX TRAVERSE(NAME,ARRAY)

PURPOSE Initializes the TRAVERSE package by setting up the catalog to be traversed and the information to be returned for each file. TRAVERSE must be called at the start of each walk, but it may not be called more than once.

INPUT

NAME An integer array of variable size containing the catalog name in FM name format. This argument need not be preserved for the remainder of the traversal.

(Word)

(0) Number of characters in the catalog name.

(1-n) Catalog name in packed character format (includes terminating !).

ARRAY A 20-word integer array that defines the details of the walk to be performed.

(Word)

(0) Option flags for tree walk:

Bit		(Leftmost bit = 0.)
0-7		Reserved
8	OFF	Return all IND values (IND is a one-word character string returned by NEXTNODE).
	ON	Return only if IND = OK or IND = ND.
9	OFF	Call the system error routine if an error occurs.
	ON	Function return any error codes.
10	OFF	Return all types of files.
	ON	Return only file types specified in ARRAY (5-20).
11	ON	Return only files dated before date in ARRAY(1-2).

12	ON	Return only files dated since date in ARRAY (3-4).
13	OFF	Do not return temporary files on IND = OK.
	ON	Return temporary files on IND = OK.
14	OFF	Execute tree walk from left to right, buttom up (See NOTE).
	ON	Execute tree walk from right to left, top down (See NOTES).
15	OFF	Traverse one level of catalog structure.
	ON	Traverse all levels of catalog structure.

(1-2) With Word 0, Bit 11 ON, BEFORE date in FM name format.

(3-4) With Word 0, Bit 12 ON, SINCE date in FM name format.

(5-20) File types to return on IND = OK.

FUNCTION RETURN

With ARRAY(0) Bit 9 set:

0	Successful TRAVERSE.
OTHER	Appropriate error code.

ERRORS

'C005'x	Catalog not found.
'C03B'x	PDL overflow.
'C045'x	No channel avalable for new level.

NOTES

- See descriptions of FREAD, ROPEN, and CLOSE for other error codes returned.
- For a description if File Manager errors, refer to SYSNEWS.ERROR.FM.
- ARRAY(0) Bit 14 ON:

  If the new entry is a catalog, push the current catalog onto the stack and process the new catalog. When the new catalog is finished; pop the stack and return the catalog as a normal entry. If the entry is not a catalog, then return the entry.

- ARRAY(0) Bit 14 OFF:

  Make two passes over each catalog. In the first pass, return each entry as encountered. In the second pass; if the new entry is a catalog, push the current catalog onto the stack and process the new catalog. If the entry is not a catalog, then ignore it.

NEXTNODE

SYNTAX NEXTNODE(IND,NAME,ARRAY)

PURPOSE Returns the next node or file in the catalog to be processed by the calling routine. Outputs the file or catalog name.

OUTPUT

IND Describes the current status of the traversal and indicates the output to NAME and ARRAY. The output to IND is a one-word literal character string, as follows:

=LP	Start of catalog.	
	NAME	Node name.
	ARRAY	ROPEN extended array.
=OK	Process this node.	
	NAME	Node name.
	ARRAY	Catalog entry for NAME.
=RP	End of catalog.	
	NAME	Node name.
	ARRAY	Blank.
=ND	End of TRAVERSE.	
	NAME	Blank.
	ARRAY	Blank.
=PT	Node is a catalog that cannot be traversed because of protection or modification. In this page IND=PT takes the place of IND=LP and IND=RP returns. IND=OK returns are not made.	
	NAME	Node name.
	ARRAY	Catalog entry for node.

ABORTRAV

SYNTAX ABORTRAV

PURPOSE Terminates the tree walk. It may be called at any point to abort a tree walk or cleanup after an error return. At the end of the traverse, (IND = ND) NEXTNODE will call ABORTRAV.

TREAD

SYNTAX	TREAD(UNIT,CNT,ARR)	
PURPOSE	Reads a physical record from magnetic tape. If necessary, it unpacks the record.	
INPUT	UNIT	Two-character task unit name for tape unit
	CNT	Maximum number of words to be read into ARR.
OUTPUT	ARR	Array of data from tape record
FUNCTION RETURN	Actual record length in words.	
NOTE	Packed records on a seven-track unit may cause more than one physical tape record to be read.	

TREADNW

SYNTAX — TREADNW(UNIT,ARGLST,ARR)

PURPOSE — Reads a physical record from magnetic tape and returns the address of the I/O flag. This is the no-wait version of TREAD.

INPUT

UNIT	Two-character task unit name for tape unit.
ARGLST	Two-word array required for the call to the tape handler.

ARGLST(1) is the maximum number of words to be read.
ARGLST(2) is set to the number of words actually read and is not valid until TESTIO returns an I/O complete.

OUTPUT

ARGLST(2)	Count of words actually read.
ARR	Array containing data from tape record.

FUNCTION RETURN — Address of IOFLAG assigned

NOTES

- Applications programs should call TESTIO before attempting to use the data.
- On a seven-track tape unit with packed records, all the advantages of NO-WAIT I/O are lost since the records have to be unpacked.
- Applications programs do not require special handling.

TSTBIT

SYNTAX TSTBIT(STRING,INDEX)

PURPOSE Returns the value of a bit.

INPUT
- STRING — Bit string array.
- INDEX — Bit index into STRING (leftmost bit is 0).

FUNCTION RETURN
- 0 — Bit is OFF.
- 1 — Bit is ON.

TSTZERO

SYNTAX TSTZERO(DBNUM)

PURPOSE Tests double integer for zero value.

INPUT
- DBNUM — Double integer to be tested.

FUNCTION
- 0 — DBNUM equals zero.
- ≠0 — DBNUM not equal to zero.

TWRITE

SYNTAX TWRITE(UNIT,CNT,ARR)

PURPOSE Writes a physical record onto magnetic tape. If necessary, the record is unpacked.

INPUT

UNIT	Two-character task unit name for tape unit.
CNT	Record length (word count)
ARR	Record data buffer

OUTPUT

ARR	Contents of the buffer.

NOTE More than one physical tape record may be written when packed records are used on a seven-track unit.

TWRITENW

SYNTAX — TWRITENW(UNIT,ARGLST,ARR)

PURPOSE — Writes a physical record on magnetic tape then, instead of waiting, it returns the I/O flag.

INPUT

UNIT	Two-character task unit name for tape unit.
ARGLST	Two-word array required for call to tape handler.
	ARGLST(1) Number of words to write.
ARR	Record data buffer.

OUTPUT

ARGLST(2)	Number of words written.

FUNCTION RETURN — Address of the I/O flag

NOTES

- The application program should call TESTIO before attempting to use the data in ARGLST.
- On a seven-track tape unit with packed records, all the advantages of NO-WAIT I/O will be lost. Once the records are packed, more than one physical record may be written for each logical record passed.
- In application programming, all calls to TWRITENW are no-wait.

TYPE

SYNTAX TYPE(CNT,LINE)

PURPOSE Outputs a line of characters to the COMDEV.

INPUT

CNT The number of characters to be output.

LINE Packed character string.

NOTES

- TYPE inserts a carriage return and a line feed at the end of the line.
- The line will also be output to the HARDEV if the hard copy toggle is on.
- TYPE calls CHKINTR after each line is output, so characters may be removed from the TIS (see CHKINTR).
- See TYPOUT and LP.

TYPEDBHX

SYNTAX TYPEDBHX(NUMBER)

PURPOSE Outputs a double integer in hexadecimal form.

INPUT NUMBER Double integer to convert for output

OUTPUT Two blanks, four hex digits, a slash, then the last four hex digits.

TYPEDBI

SYNTAX TYPEDBI(NUMBER)

PURPOSE Outputs a double precision integer in decimal form.

INPUT NUMBER Double precision integer.

OUTPUT The number is preceded by two blanks. Leading zeros are suppressed.

TYPEHEX

SYNTAX TYPEHEX(NUMBER)

PURPOSE Outputs an integer in hexadecimal form.

INPUT NUMBER 16-bit integer.

OUTPUT Four hex digits — leading zeros are not suppressed.

TYPEINT

SYNTAX TYPEINT(NUMBER)

PURPOSE Outputs an integer in decimal format.

INPUT NUMBER Integer.

OUTPUT NUMBER Is right-justified, padded with blanks, and output in a six-character field (equivalent to a FORTRAN I6 format).

TYPEOK

SYNTAX TYPEOK(FILENAME,MESSG1,MESSG2)

PURPOSE Tests for OK response to a message printed on the COMDEV. The response may come from the COMDEV or from an execute file.

Special Feature

If the response is QUIT, then TYPEOK does a LBLGO to −3. This is identical to an INTR-QUIT <ESC>Q.

INPUT

FILENAME Filename in standard FMNAME format, or a third message.

MESSG1 First message:

(1) Character count.
(2-n) Message characters.

MESSG2 Question asked 'TYPE OK TO' MESSG2:

(1) Character count.
(2-n) Message characters.

FUNCTION RETURN

0 Response was OK.
−1 No OK response — COMDEV may have been detached.

NOTE If a third message is used and the final character in the message is (!), the (!) will not be printed (as in a filename).

TYPIN

SYNTAX TYPIN(MAX,BUF,CNT)

PURPOSE Inputs a line of characters from the COMDEV or an execute file.

INPUT
MAX Maximum number of characters to be input.

OUTPUT
BUF Destination Array for line of characters.

CNT Actual character count (CNT $\leq$ MAX).

NOTES

- TYPIN inputs characters until a carriage return is found, places the line in BUF and returns the actual character count in CNT. If MAX characters are input before a carriage return, the line is considered complete.
- Unless an execute file is active, input is read from the COMDEV.
- Any special system characters in the input will be processed as they are received. The line stored in BUF will reflect their actions even though the characters themselves will not be stored. A line may be restarted, a character deleted, and so forth. HARDCOPY and NOPRINT toggle characters are processed in the same way.
- See also STYPIN, TYPEOK, TYPOUT.

TYPOUT

SYNTAX TYPOUT(CNT,STRING)

PURPOSE Outputs a string of characters to the COMDEV (and to the HARDEV if the toggle is on).

INPUT
CNT Number of characters to be output.

STRING Packed character string.

NOTES

- TYPOUT does not add a carriage return or line feed at the end of the line; the cursor is left after the last character typed.
- See also, TYPE.

UNATTACH

SYNTAX UNATTACH(NAME)

PURPOSE Detaches a unit from a task.

INPUT NAME Two-character (1 word) name assigned to the unit.
−1 to detach all units.

FUNCTION RETURN
1 Successful completion.
0 The name is bad, or COMDEV is released and there is another unit named 'SD'.

WAITIO

SYNTAX WAITIO(IOFLAG)

PURPOSE Changes a NOWAIT I/O request into a WAIT I/O request.

INPUT IOFLAG Address of a valid IOFLAG.

OUTPUT Error messages:

F005 Called with invalid IOFLAG address.
F006 Error was generated during the I/O request.

NOTE WAITIO resets the IOFLAG value. This eliminates any need to call TESTIO(IOFLAG).

THE WRITBFIL UTILITY PACKAGE

The general sequence for writing to a binary sequential file with byte-oriented counts would be to open with WRITFILE, write blocks of words with WRITBYTE, and close with WRITCLOS.

WRITBFIL

SYNTAX

WRITBFIL(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE

Opens a binary sequential file for writing with byte-oriented counts.

INPUT

FSB(11)	File Status Block. FSB(1) must equal one for the first call buffer.		
BUF	Disc buffer.		
BUFSIZ	Size of BUF (sectors).		
FNAME	Filename:		
	FNAME(1)		Byte count of name, including final (!).
	FNAME(2-N)		Filename with (!).
OPTIONS	Bit		(Leftmost bit is 0.)
	7	ON	Create new catalogs as needed.
		OFF	Error if catalog level is missing.

WRITBYTE

SYNTAX WRITBYTE(FSB,BUF,COUNT,BLOCK,OFFSET)

PURPOSE Writes a record to an open binary file (see WRITBFIL).

INPUT

FSB(11)	File Status Block.
BUF	Disc buffer.
COUNT	Number of bytes in the record.
BLOCK	Array containing the record to be written.
OFFSET	Byte offset in the array (counted from 0).

WRITBCLS

SYNTAX WRITBCLS(FSB,BUF)

PURPOSE Closes a binary sequential file opened with WRITBFIL.

INPUT

FSB(11)	File Status Block.
BUF	Disc buffer.

THE WRITFILE UTILITY PACKAGE

The general sequence for writing to a binary sequential file would be to open with WRITFILE, write blocks of words with WRITBLOK, and close with WRITCLOS.

These three routines can also be used to write a random access binary file. There are three functions available for using the sequential access pointer of a random access file (see Section 5, The READFILE Utility). A call to READMARK reads the position of the pointer, while READPOS and READTOP can be used to move the pointer.

An existing file may be modified in place by using MODIFILE to open the file. READBLOK, WRITBLOK, READPOS, READMARK, and READTOP can all be used to change data in the file. To close it, use WRITCLOS.

WRITFILE

SYNTAX WRITFILE(FSB,BUF,BUFSIZ,FNAME,OPTIONS)

PURPOSE Opens a binary sequential file for output.

INPUT

FSB File Status Block.

BUF Disc buffer.

BUFSIZ Size of disc buffer in sectors.

FNAME Filename in FM format:

Word 1	Character count (including !).
Word 2-n	Filename (terminated by !).

OPTIONS(1)

Bit		(Leftmost bit is 0.)
2	ON	Initial number of sectors to allocate is in OPTIONS(2-3).
	OFF	Allocate default number of sectors.
3	ON	Force the file to be contiguous.
	OFF	Non-contiguous sectors may be allocated.

INPUT (Continued)

OPTIONS(1)	7	ON	Create new catalogs as required.
		OFF	Generate error if all catalogs do not exist.
	15	ON	Error condition results in error FLAG returned as function name.
		OFF	Error results in the call ERROR(FLAG).
OPTIONS(2-3)			Number of sectors to allocate when file is opened (only if bit 2 is ON).

FUNCTION RETURN

OPTIONS(1) bit 15 ON:

0	Indicates no error
FLAG	Indicates type of FILE MANAGER ERROR

ERRORS

For description of FILE MANAGER ERRORS, refer to: SYSNEWS.ERROR.FM.

NOTES

- The FSB (File Status Block) is described in LIB.FM.READFILE.
- No routines in the READFILE family contain local variables to maintain. Only the integrity of FSB and BUF is important.

WRITBLOK

SYNTAX WRITBLOK(FSB,BUF,COUNT,BLOCK)

PURPOSE Writes, or modifies, a block of words in a binary sequential file.

INPUT

FSB(11)	File Status Block.
BUF	Disc buffer.
COUNT	Number of words to write.
BLOCK	Array of words to write.

NOTE Before WRITBLOK is called, the file must be opened with MODIFILE or WRITFILE.

WRITCLOS

SYNTAX WRITCLOS(FSB,BUF,OPTIONS)

PURPOSE Closes a binary sequential file.

INPUT

FSB File Status Block.

BUF Disc buffer.

OPTIONS(1)

Bit		
0	ON	If open for create, delete the file. If the file is open for create with supersede, do not supersede the file. Instead, delete the new file and preserve the original).
1	ON	If bit 0 is OFF, change user attribute values to OPTIONS(2) and OPTIONS(3).
2	ON	Do not update the access date.
3	ON	If bit 0 is OFF, change the file size the value specified in OPTIONS(4-5).
4	ON	If bit 0 is OFF, change the creation date and time to values specified in OPTIONS(6-7).
5	ON	If bit 0 is OFF, change the system attibute words to OPTIONS(7-8).
15	ON	Routine function returns errors generated by CLOSE.
	OFF	Errors are processed by the system error handler.

OPTIONS(2-3) User attribute values (if bit 1 on).

OPTIONS(4-5) Total sector count for file (if bit 3 is ON).

OPTIONS(6-7) Creation time and date for file (if bit 4 is ON).

OPTIONS(8-9) System attribute words (if bit 5 is ON).

XEQTCOMM

SYNTAX XEQTCOMM(COUNT,COMMAND)

PURPOSE Executes a system command passed as a subroutine argument.

INPUT

COUNT	Character count of command string.
COMMAND	Command string to be executed.

NOTE This routine does not return to the calling code after invoking the system command.

Section 6
SYSTEM FORMATS

Section 6
SYSTEM FORMATS

This section describes:

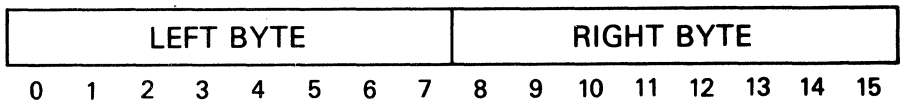
- Common system data structures.
- File formats.
- Data types and how they are represented in CGOS 200.

BIT STRING FORMAT

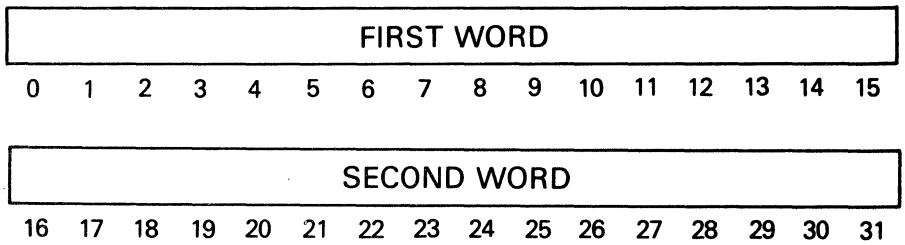
Bits are generally numbered from left to right, starting with bit 0. This is true for:

- All CGOS 200 manuals.
- All CGOS 200 subroutine descriptions, code and data structures.
- The front panels and backplanes of all Computervision CPUs (CGP-100) and the CGP-100 maintenance panel.

SINGLE WORD BIT NUMBERING:



DOUBLE WORD BIT NUMBERING:

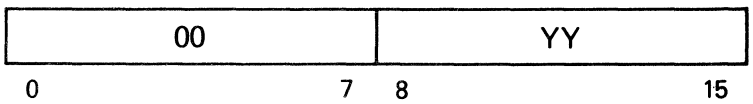


CAUTION

Bits are numbered from right to left in CADDS application routines. Numbering schemes may vary for peripherals that interface with Computervision systems.

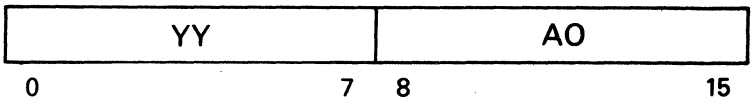
BYTE FORMAT

For the byte YY hexadecimal:



CHARACTER FORMAT

For the character whose ASCII code is YY hexadecimal:

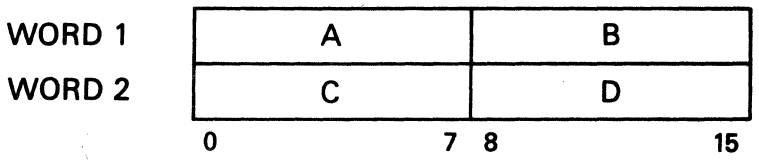


A0 hexadecimal is the byte for the blank ASCII character marked parity.

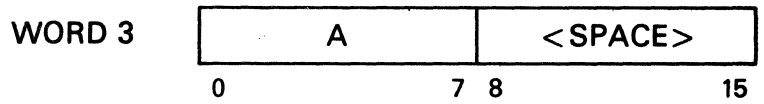
Character String Format

Character strings are packed two characters per-word starting with the left byte of the first word. If the string has an odd number of characters, it is left-justified and padded with a blank (ASCII code A0 hexadecimal).

The character string 'ABCD' is stored as:



The character "A" is stored as:



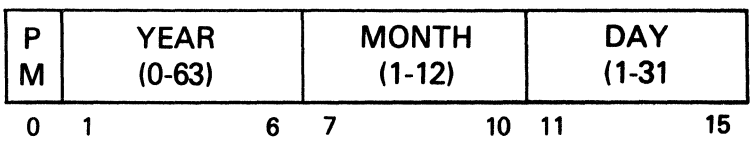
NOTE

All characters are stored as seven-bit ASCII codes with the most significant eighth bit set. BCD codes are never used, even when explicitly mentioned in the documentation.

DATE AND TIME FORMATS

SYSTEM (FILE MANAGER) DATE AND TIME FORMAT

DATE:



where PM is AM/PM indicator, ON for PM, OFF for AM, and Year is number of years since 1960.

TIME:

SECONDS (0-43199)

0 15

where Seconds is the number of seconds in a half-day.

INTERNAL TIME FORMAT

CLOCK (1):

YEAR RIGHT 2 DIGITS	DAY (1-366)

0 15

0 15

CLOCK (1):	MINUS TICKS LEFT
CLOCK (2):	UNTIL MIDNIGHT

16 31

(1 tick = 10 milliseconds)

EXTERNAL TIME FORMAT

0 15

TIME (1):	YEAR RIGHT 2 DIGITS
TIME (1):	DAY (1-366)
TIME (2):	MINUTES
TIME (3):	TENTHS OF SECONDS

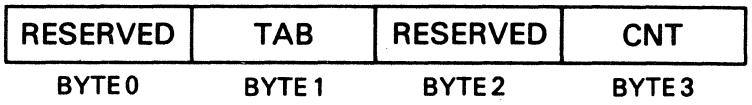
TEXT FILE FORMAT

Under the File Management catalog structure:

Word	*Contents*
(1-2)	Double word indicating number of sectors in file.
(3)	Number of bytes of useful information in last sector of file.
(4)	0.
(5-N)	Line data for file.

Each line consists of a four-byte header followed by the N characters from that line.

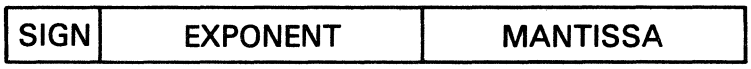
HEADER:



TAB Number of blanks to insert in front of text.
CNT Number of bytes of text in this line.

Floating Point Format

Floating Point (Real) Format provides an approximate representation for all numbers within a very wide range. Allocation of more bits per number (greater PRECISION) increases accuracy by decreasing the difference between consecutive numbers. Three fields of bits are used to represent a number in Floating Point Format:



Sign

SIGN is a one-bit field. If the bit value is one, the floating Point number is negative. If the bit is zero, then the number is positive.

Exponent

An eight-bit field where values are stored in two's complement binary with an inverted sign bit. This form of storage is often called "excess 128" as the result of adding 128 to the normal two's complement binary value.

EXPONENT is the power of two by which the fraction in the MANTISSA should be multiplied to reconstruct the floating point number. Thus, the pattern '80'x within the field (which has value of 0) means that the (fractional) MANTISSA should be multiplied by 2**0, or 1; i.e., the mantissa is the represented (fractional) number. Similarly, patterns of '81'x or more (values > 0) imply multiplication by powers of two (left shift) during reconstruction.

Note

Two asterisks (**) mean "raised to the power of".

Mantissa

A variable length field, 23 bits in SINGLE precision format or 55 bits in DOUBLE precision format. MANTISSA stores the magnitude of the fraction thAT reconstructs the floating point value when multiplied by EXPONENT. For accuracy, the EXPONENT and MANTISSA are adjusted until the fraction represented by the MANTISSA falls into the range:

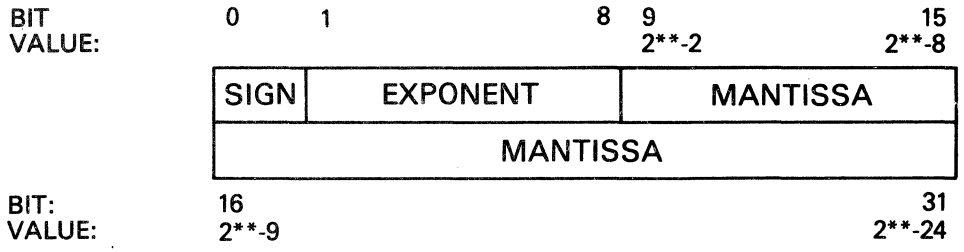
$$1/2 < \text{fraction} < 1$$

This is called "normalizing" the fraction. For a normalized fraction represented in binary, the first bit to the right of the binary point is the most significant bit, and its value is always one. To extend precision, this bit is NOT explicitly stored. The most significant stored bit in the mantissa field is actually the bit representing 1/4; thus the 1/2 bit is assumed to equal one. This allows the mantissa to store 24 significant bits for SINGLE precision, and 56 significant bits in DOUBLE precision. This hidden bit is called a "phantom bit representation" for the mantissa.

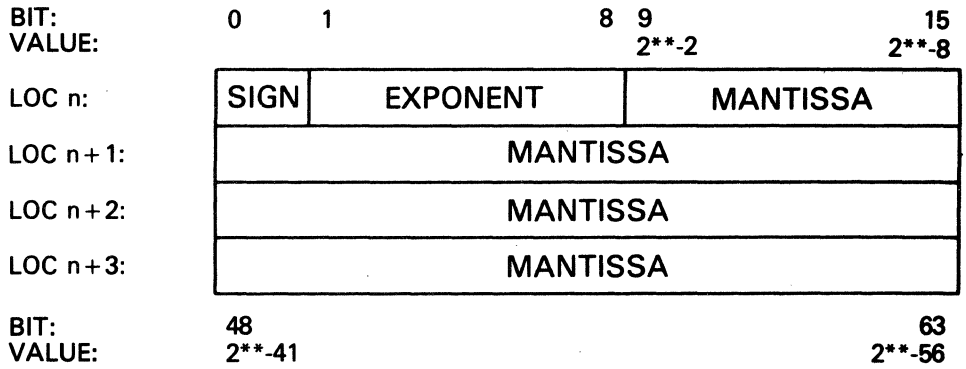
STORAGE FORMATS

SINGLE precision floating point format allocates two memory words, while DOUBLE precision allocates four words. In either precision, the first memory word contains the SIGN, EXPONENT, and most significant bits of MANTISSA fields:

Single Precision



Double Precision



Zero

Zero is the only number not normally represented in the above formats; it has no most significant bit to be normalized. In either SINGLE or DOUBLE precision, the number in the EXPONENT and MANTISSA is true zero (0.0), rather than the smallest possible nonzero number. Calculations which produce values less than or equal to the smallest possible nonzero number are considered to have produced a zero (0.0) instead. Similarly, minus zero (sign bit being the only nonzero bit) is converted to true zero (all bits zero).

Range

In absolute value, a floating point number has the following ranges:

SINGLE Precision:

1/2 x 2**-128 to (1 - 2**-24) x 2**127

DOUBLE Precision:

1/2 x 2**-128 to (1 - 2**-56) x 2**127

This is approximately 10**-38 to 10**38.

Accuracy

SINGLE Precision:

1 part in 2**24 (7 decimal digits)

DOUBLE Precision:

1 part in 2**56 (16 decimal digits)

INTEGER FORMAT

Each number is represented exactly in INTEGER format. The difference between consecutively represented numbers is ONE (unity), and thus independent of PRECISION. Allocation of more bits per number extends the range of representable numbers, but not the accuracy of each number.

Standard two's complement notation for INTEGERS allows both positive and negative numbers to be represented as 16 bits (SINGLE) or 32 bits (DOUBLE) precision (these correspond to one word or two words, respectively).

SINGLE integer range:

−32,768 to 32,767 inclusive

DOUBLE integer range:

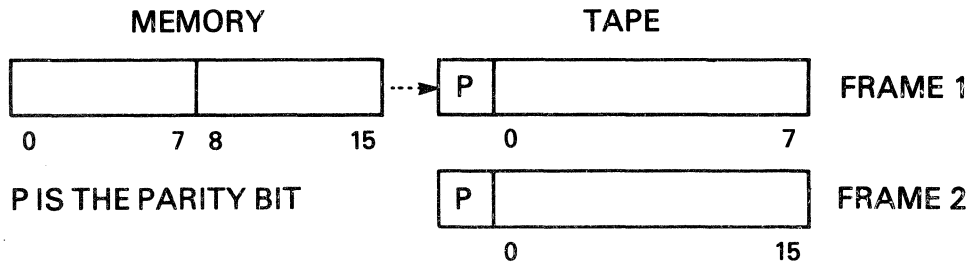
−2,147,483,648 to 2,147,483,647 inclusive

MAGNETIC TAPE FORMAT

Tape controllers read or write up to '2000'hex frames as a single record. This corresponds to '1000'hex words on 9-track tape. High level system tape I/O routines recognize this and allow transfers of '1000'hex words per call.

9-Track Tape

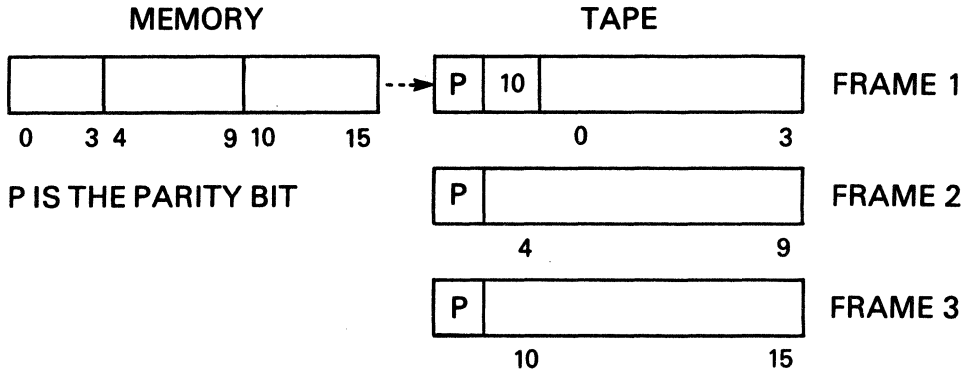
Two frames of tape store one CGP-100 16-bit word. Each frame consists of a parity bit followed by eight bits of data. The word is converted so that bits 0-7 are stored in the first frame and bits 1-15 are stored in the second frame.



7-Track Tape

PACKED FORMAT

Packed and unpacked formats are used. Machine words cannot be evenly divided into tape frames on 7-track tape, so Packed Format is used. Each 7-track tape frame consists of a parity bit followed by 6 bits of data. One 16 bit machine word is "packed" into 3 tape frames. Frame 1 of the tape has a binary 10 in the first two bits followed by the data from memory word bits 0-3; frame 2 contains bits 4-9 from the memory word; and frame 3 contains bits 10-15.

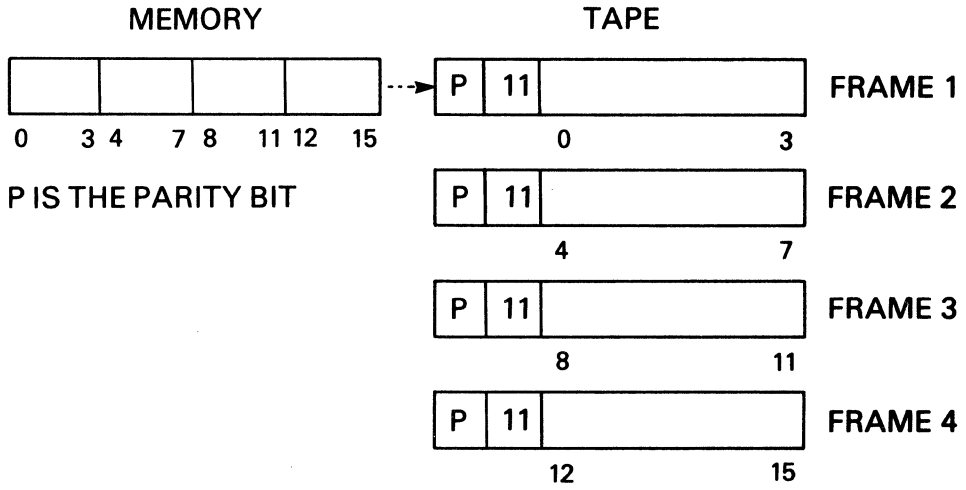


Since memory words require three frames on 7-track tape, the entire transfer may require more than one tape record. To resolve this problem, the system software splits the complete request into multiple tape records. These records are at most '400'hex words long and are begun with a three word header (9 frames). The header format is:

Word 0: Physical record number for this logical record ($\geq$ 1).
Word 1: Total number of physical records in this logical record ($\geq$ 1).
Word 2: Maximum physical record size ('400'hex)

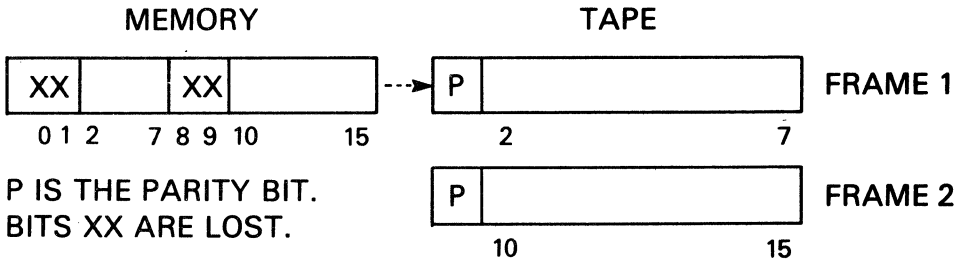
OLD PACKED FORMAT

The system supports Old Packed format to allow reading of tapes prepared by earlier CADDS O/Ss. Old Packed format is only supported for reading tapes. It 'packs' each memory word into four tape frames, with four memory word bits per frame. The first two bits of each frame contain the binary value 11. Each tape record begins with a twelve-frame header in the same form as the header for Packed Format.



UNPACKED FORMAT

This format transfers to tape exactly like 9-track format, two frames per word. Unfortunately, 7-track tape has only six bits per frame, so four bits (0-1 and 8-9) are lost from each memory word.



INDEX

Index

ABORTRAV 4-4, 5-171, 5-172, 5-175
ACCURACY 6-7
ADD 2-24
ADDCAT 2-24
Address space 3-5
ADJUST 4-10, 5-5
ALLOPG 4-4, 5-6
AND 2-11
Application bit numbering 6-1
Application debugger 3-4
Array indexing 5-2
Arrays 2-2
ASCII 6-2
ATTACH 4-6, 5-6

Bit mask 2-25
Bit numbering 5-2, 6-1
Block 2-8
Boolean expressions(loader) 2-10
BREAK 3-6
Breakpoints 3-5
BRIEF 2-17
Byte format 6-1

CADDS ..... 1-1, 2-1, 2-31 thru 2-35
CADDS overlays 2-1,2-31
CARDIN 4-6, 5-7
CATBREAK 4-4, 5-8
CATWALK 4-4, 5-9
Character format 6-2
Character string format 6-2
CHKBREAK 4-9, 5-13
CHKINTR 4-9, 5-14
CHKPROT 4-2, 5-4, 5-15
CHKQUIT 4-9, 5-16
CHKSTOP 4-9, 5-17
CHKSUM 4-13, 5-17
CHKTRACE 3-2, 4-7, 5-18
CHKUPROT 4-2, 5-4, 5-18
CKPPTD 4-6, 5-19
CKRPTD 4-6, 5-19

Index (Continued)

CLEARCOM 4-5, 5-19
CLOSE 4-2, 5-20
CLRBIT 4-12, 5-21
CLRBREAK 4-9, 5-22
CMBYTF 2-2, 4-10, 5-22
CMBYTT 2-2, 4-10, 5-23
CMPDAT 4-5, 5-23
COMDEV 5-2
Command execution 2-36
Command table search 2-36
COMMTAB 2-36, 2-37
Compiler 2-1
Compiler options 2-3
Compiling routines 2-3
COMPN 4-10, 5-24
COMPNAM 4-10, 5-25
COMPUS 4-10, 5-25
CONDITION 2-10
CONTROL-D 3-4
CONTROL-T 3-4
COPEN 4-2, 5-4, 5-26
COPYFILE 5-31
CORE image 2-18
CORELOAD 2-18
Coreload 2-7
CORORG 2-8
COUNT 3-7
Cross-referenced commands 2-37
CROSSREF 2-30
CVSCOMMAND 2-36
CWRITE 2-9, 3-2

Date format 6-2
DBADD 4-10, 5-32
DBCMPR 4-10, 5-32
DBDCR 4-10, 5-33
DBDIV 4-10, 5-33
DBHEXBCD 4-11, 5-35
DBHEXLST 5-34
DBHXLST 4-7

Index (Continued)

DBHXNM 4-7, 5-35
DBINC 4-10, 5-34
DBINT 4-7, 5-36
DBINTBCD 4-11, 5-37
DBINTLST 4-7, 5-37
DBLE 4-11, 5-38
DBLSH 4-10, 5-38
DBMAX 4-10, 5-39
DBMIN 4-10, 5-39
DBMUL 4-10, 5-40
DBNEG 4-10, 5-40
DBRSH 4-10, 5-41
DBSUB 4-10, 5-41
DEBUGGER access 3-4
Debugger commands 3-6
Debugger syntax 3-4
DELETE 4-2, 5-4, 5-42
DELETEXT 4-4, 5-43
DELFIL 4-4, 5-44
DFLOAT 4-11, 5-44
DFLOATL 4-11, 5-45
DIVUS 4-10, 5-45
Double precision 6-5
DUMP 2-9, 3-7

EDITCMTB 2-36
Editor FC command 2-1
ELAPTIME 5-46
ELSE 2-9
END 2-9
END-OF-BLOCK 2-8
ENDC 2-9
EQU 2-9
ERROR 4-6, 5-47
EXECUTE 2-8
Explicit references 2-30
EXPONENT 6-4
External routines 5-1
External time format 6-3

Index (Continued)

File management 5-4
File management errors 5-4
File status block 5-3
File utility packages 5-3
FILENAME 2-9, 2-10
FILL 4-11, 5-48
FILLBYTT 4-11, 5-48
FILLCHRT 4-11, 5-49
FILLDB 4-11, 5-49
FILLF 4-11, 5-50
FLBCD 4-11, 5-50
Floating point format 6-4
FLOATL 4-11, 5-51
FM name format 5-3
FMCNTRNM 4-4, 5-51
FMEXPNM 4-4, 5-52
FMIDAT 4-5, 5-53
FMIDNF 4-8, 5-54
FMIDNT 4-8, 5-55
FMNAME 4-8, 5-56
FMNLST 5-56
FMTDAT 4-5, 5-57
FNDVAL 5-58
FORTRAN 2-2
FREAD 4-2, 5-4, 5-59
FREEPG 4-4, 5-60
FSB 5-3
Function 5-1
FUNCTION RETURN 5-1, 5-2
FWRITE 4-2, 5-4, 5-61

GENCOM 2-26
GENINDX 2-28
GETBIT 4-12, 5-63
GETCLEAR 4-2, 5-70
GETCLOSE 4-2, 5-72
GETDAT 4-5, 5-84
GETFILE 4-2, 5-4, 5-64
GETFLD 4-12, 5-73
GETLABEL 4-9, 5-74

INDEX (Continued)

GETLINE....4-2, 5-68
GETLINEB....4-2, 5-69
GETMARK....4-2, 5-71
GETPG....4-4, 5-75
GETPGP....4-4, 5-76
GETPOS....4-2, 5-70
GETPPTD....4-6, 5-77
GETPUNCH....4-6, 5-78
GETRPTD....4-6, 5-79
GETSTART....4-2, 5-71
GETSTAT....4-13, 5-80
GETTASK....4-5, 5-81
GETTASKF....4-5, 5-81
GLOBAL....2-10, 2-11, 2-18
Globals....2-30
GLP....4-8, 5-82

HARDEV....5-2
HEAPSORT....4-13, 5-83
HELP....1-3, 1-4
Hex numbers....2-8
HEXBCD....4-11, 5-83
HEXDMP....4-13, 5-84
HEXLST....4-7, 5-85
HEXNUM....4-7, 5-86
HIBERN8....4-5, 5-86

IDENT....4-7, 5-87
IF....2-10
IFIXD....4-11, 5-87
INCLUDE....2-26
INITCHAR....4-7, 5-88
INITNM....4-8, 5-88
INPUT....5-1

INSERT....2-11, 2-18, 2-25, 2-30, 2-31
Insert....2-18
Insert file....2-18
INSERT.SYMFILE.STANDARD....2-31
INSESORT....4-13, 5-89

Index (Continued)

INSESRTN 4-13, 5-90
INT 4-7, 5-91
INTBCD 4-11, 5-91
Integer format 6-7
Internal time format 6-3
INTLST 4-7, 5-92
IPDL 4-12, 5-93
ISNGL 4-11, 5-93

JANMAKE.SYM.BASIC 2-31, 5-12
JANMAKE.SYM.DBLINT 2-31
JANMAKE.SYM.FLPT 2-31
JANMAKE.SYM.FMSYS 2-31, 5-4
JANMAKE.SYM.SYS 2-31
JANMAKE.SYM.SYSOV 2-31
JANMAKE.SYMFILE.SYSFM 2-31
JANMAKE.SYMFILE.SYSFMLI 2-31
JANMAKE.SYMFILE.SYSOV 2-31
JANMAKE.SYMFILE.SYSOVFM 2-31
JANMAKE.SYMFILE.SYSOVFMFP 2-31
JANMAKE.SYMFILE.SYSOVFMLI 2-31
JANMAKE.SYMFILE.SYSOVFMLIFP 2-31
JANMAKE.SYMFILE.SYSOVFP 2-31

LABS 4-10, 5-94
LAND 2-11
LBLGO 5-94
LBLSET 2-2, 5-95
LDBYTF 2-2, 5-95
LDBYTT 2-2, 5-96
LDCHRF 2-2, 5-97
LDCHRT 2-2, 5-97
LFIX 5-98
LFIXD 5-98
LIB 2-12, 2-26
LILL 2-26
Linking code 2-7
LIST 2-17
LISTIN 2-17
LISTLOAD 2-26
LISTLOAD commands 2-26

Index (Continued)

LISTSYM 2-12
LOAD 2-1, 2-7, 2-13, 2-15, 2-26
Load library 2-24
Loader diagnostic errors 2-19
Loader errors 2-18
Loader inserts 2-30
Loader major errors 2-20
Loader minor errors 2-19
Loading code 2-7
LOADLIB 2-12
Loadlib 2-24
LOADLIB.FORTLIB 2-2, 2-31, 4-1
LOADLIB.OSLIB 2-31, 3-3, 4-1
LOADLIB.OSLIB3 4-1
LOADSYM 2-13, 2-25, 2-31
Local variables 3-5, 3-8
LP 4-8, 5-99

Magnetic tape format 6-7
MAKE 2-15, 2-25
Make 2-1, 2-8, 2-12, 2-18
Make file 2-7, 2-18
MANTISSA 6-4
MAPONLY 2-18
MAX 2-11
MAXIMUM 4-10, 5-100
Memory pages 4-4
MINIMUM 4-10, 5-100
MODIFILE 4-3, 5-101
MOPEN 4-2, 5-105
MOV 4-12, 5-109
MOVB 4-12, 5-109
MOVD 4-12, 5-110
MOVEWORD 4-12, 5-110
MOVF 4-12, 5-111
MOVL 4-12, 5-111
MRGCOPY 2-28
MVBYTF 2-2, 4-12, 5-112
MVBYTT 2-2, 4-12, 5-112

Index (Continued)

NEWCAT 2-16, 2-17
NEXTNODE 4-4, 5-171, 5-174
Nine-track tape 6-7
NLEV 2-29
NLISTSYM 2-12
Nodes 5-171
NOFILE 2-17
NOSEP 2-12
NOT 2-11
NOTRACE 3-2
NXTCHAR 4-7, 5-113
NXTFPG 5-114
NXTNAM 4-8, 5-114

Object 2-18
Object file 2-18
OCTBCD 4-11, 5-115
Old packed format 6-8
On-line documentation 1-2
Operators (loader) 2-10
OPTIONS 5-3
OR 2-11
OREAD 3-3, 4-2, 5-115
OUTPUT 5-1
OVCAT 2-17
Overlay 2-7, 2-18, 3-3
OVWRITE 2-8, 2-16

Packed format 6-7
PAGFIL 5-116
PERROR 4-6, 5-117
PNCHLDR 4-6, 5-119
POP 4-12, 5-119
POPN 4-12, 5-120
PPT 4-6, 5-120
PPT1 ..... 4-6, 5-121
PPTN 4-6, 5-121
PRINT 2-13
PSTAT 2-28, 2-29
PUSH 4-12, 5-122

Index (Continued)

PUSHN 4-12, 5-122
PUTABORT 4-3, 5-126
PUTBIT 4-12, 5-122
PUTCLEAR 4-3, 5-126
PUTCLOSE 4-3, 5-127
PUTFILE 4-3, 5-3, 5-123
PUTFLD 4-12, 5-128
PUTLABEL 4-9, 5-129
PUTLINE 4-3, 5-124
PUTLINEB 4-3, 5-125

RANGE 6-7
READBCLS 4-3, 5-134
READBFIL 4-3, 5-130
READBLOK 4-3, 5-139
READBYTE 4-3, 5-134
READCLOS 4-3, 5-135
READFILE 4-3, 5-140
READMARK 4-3, 5-140
READPOS 4-3, 5-141
READSECT 4-3, 5-139
READTOP 4-3, 5-141
REDEF 2-14
RENAME 4-2, 5-5, 5-142
RESET 3-7
RESETLBL 4-9, 5-146
RETERR 4-6, 5-146
ROPEN 4-2, 5-5, 5-147
RPT 4-6, 5-151
RPT1 ..... 4-6, 5-151
RPTN 4-6, 5-152
RUN 2-1, 2-15, 2-17
RUN PROG 2-1, 2-2

SAVESYM 2-13, 2-14, 2-16
SELESORT 4-13, 5-153
SEP 2-14
SET 2-14
SETBIT 4-12
SETBREAK 4-9, 5-154

Index (Continued)

SETCOM 5-155
SETPG 4-4, 5-156
SETPGP 4-4, 5-157
SETPUNCH 4-6, 5-158
SETSTRG 4-5, 5-158
Seven-track tape 6-8
SHELSRTN 4-13, 5-159
SIGN 6-4
Single precision 6-5
SIZE 2-10, 2-11
SNGL 4-11, 5-160
SORT 2-28, 2-29
Source 2-18
SPID 4-4
SRTESTIO 4-9, 5-160
SRWAITIO 4-9, 5-161
STBYTF 2-2, 4-12, 5-161
STBYTT 2-2, 4-12, 5-162
STCHRF 4-12, 5-162
STCHRT 4-12, 5-162
Storage formats 6-8
STYPEOK 4-8, 5-163
STYPIN 4-8, 5-164
Subroutine trace 3-1
SUBTRACE 3-2, 4-7, 5-165
Symbol files 3-3
Symbol map 2-1, 2-5
Symbolic references 2-30
SYMFILE 2-31
SYNTAX 5-1
SYSCATLG 2-16, 2-28, 2-29
SYSCMTB 2-36
SYSCOMMAND 2-36
SYSOVLY 4-1
System calls 5-5
System library routines 4-1
System overlay routines 4-1
System references 2-30
System resident routines 4-1
SYSUSERCMTBXXXX 2-36

Index (Continued)

TAPE 4-6, 5-166
TAPENW 4-6, 5-168
TEST 2-1
TESTIO 4-9, 5-169
TESTTAPE 4-6, 5-169
Text editor 2-1
Text file format 6-4
TIME 4-5, 5-170
Time format 6-2
TOGGLE 4-8, 5-170
TOGGLES 3-5
TRACE 3-1
Trace map 3-1
Trace output 3-3
Transfer vector 2-14, 2-15
TRAVERSE 4-4, 5-171, 5-172
TREAD 4-6, 5-176
TREADNW 4-6, 5-177
Tree walk 5-171
TSTBIT 4-12, 5-178
TSTZERO 4-10, 5-178
TV 2-14, 2-18
TWRITE 4-6, 5-179
TWRITENW 4-6, 5-180
TYPE 4-8, 5-181
TYPEDBHX 4-9, 5-182
TYPEDBI 4-8, 5-182
TYPEHEX 4-9, 5-183
TYPEINT 4-8, 5-183
TYPEOK 4-8, 5-184
TYPIN 4-8, 5-185
TYPOUT 4-8, 5-185

UNATTACH 4-6, 5-186
UNDEF 2-15
UNDEFP 2-11
Unpacked format 6-9
UNUSEDP 2-11
User address space 4-4
USERCMTB 2-36

Index (Continued)

WAITIO..... 4-9, 5-187
WRITBCLS..... 4-4, 5-189
WRITBFIL..... 4-4, 5-3, 5-188
WRITBLOK..... 4-3, 5-192
WRITBYTE..... 4-4, 5-189
WRITCLOS..... 4-3, 5-193
WRITE..... 2-8, 2-10, 2-15
WRITFILE..... 4-3, 5-3, 5-190

XEQTCOMM..... 4-13, 5-194

Please cut along this line

REMARKS FORM

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications. All comments and suggestions become the property of Computervision.

TITLE: _____

Order No.: _____

TECHNICAL or EDITORIAL ERRORS (include page number):

SUGGESTIONS FOR IMPROVEMENT:

FROM:
(Please print)

NAME: _____ DATE _____

TITLE: _____

COMPANY NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 77 WOBURN, MA 01801

POSTAGE WILL BE PAID BY ADDRESSEE

Computervision Corporation
100 Commerce Way
Woburn, Massachusetts 01801

ATTN: TECHNICAL PUBLICATIONS

Fold here

Fold here

Please cut along this line

Computervision Corporation 201 Burlington Road, Bedford, Massachusetts 01730