SERIES 60 (LEVEL 6)

GCOS 6 COMMANDS ADDENDUM A

SUBJECT

Index to the Manual

SPECIAL INSTRUCTIONS

Insert the attached pages (see Collating Instructions) into Revision 1 of the manual dated June 1978.

Note:

Insert this title page behind the manual cover to indicate update of the manual with Addendum A.

SOFTWARE SUPPORTED

This update supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 Operating System. See the Manual Directory of the latest *GCOS 6 MOD 400 System Concepts* manual (Order No. CB20) for information as to later releases supported by this manual.

ORDER NUMBER

CB02A, Rev. 1

July 1978

21357
3778
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
—	i-1 through i-8, at end of manual

File No.: 1S13

7/78
CB02A

SERIES 60 (LEVEL 6)

GCOS 6 COMMANDS

SUBJECT

Detailed Description of Series 60 (Level 6) GCOS 6 Command Language

SOFTWARE SUPPORTED

This publication supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 Operating System; see the Manual Directory of the latest GCOS 6 MOD 400 System Concepts manual (Order No. CB20) for information as to later releases supported by this manual.

ORDER NUMBER

CB02, Rev. 1

June 1978

Honeywell

Preface

This manual describes the GCOS 6 command language. Unless stated otherwise, the term GCOS refers to the GCOS 6 software; the term Level 6 refers to the Series 60 (Level 6) hardware on which the software executes.

Section 1 of the manual provides an introduction to the command language. It summarizes the commands according to function, describes the command line format, explains the development of File System pathnames, and covers the use of the Break key in interrupting command execution.

Section 2 describes the format and function of each command. For ease of reference, the commands are presented in alphabetical order.

Appendix A contains detailed information that pertains to a limited number of commands. It describes the use of additional command line arguments, discusses terminal characteristics at login, and defines the pathname colon convention.

Appendix B describes the directives used with the Intersystem Link (ISL) command.

Appendix C describes the directives used with the File Change command.

Appendix D defines the standard GCOS 6 character set and its hexadecimal equivalents.

File No.: 1S13 CB02

MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set. The Manual Directory in the latest *GCOS 6 MOD 400 Systems Concepts* manual (Order No. CB20) lists the current revision number and addenda (if any) for each manual in the set.

Order No.	Manual Title
CB01	GCOS 6 Program Preparation
CB02	GCOS 6 Commands
CB03	GCOS 6 Communications Processing
CB04	GCOS 6 Sort/Merge
CB05	GCOS 6 Data File Organizations and Formats
CB06	GCOS 6 System Messages
CB07	GCOS 6 Assembly Language Reference
CB08	GCOS 6 System Service Macro Calls
CB09	GCOS 6 RPG Reference
CB10	GCOS 6 Intermediate COBOL Reference
CB20	GCOS 6 MOD 400 System Concepts
CB21	GCOS 6 MOD 400 Program Execution and Checkout
CB22	GCOS 6 MOD 400 Programmer's Guide
CB23	GCOS 6 MOD 400 System Building
CB24	GCOS 6 MOD 400 Operator's Guide
CB25	GCOS 6 MOD 400 FORTRAN Reference
CB26	GCOS 6 MOD 400 Entry-Level COBOL Reference
CB27	GCOS 6 MOD 400 Programmer's Pocket Guide
CB28	GCOS 6 MOD 400 Master Index
CB30	Remote Batch Facility User's Guide
CB31	Data Entry Facility User's Guide
CB32	Data Entry Facility Operator's Quick Reference Guide
CB33	Level 6/Level 6 File Transmission Facility User's Guide
CB34	Level 6/Level 62 File Transmission Facility User's Guide
CB35	Level 6/Level 64 (Native) File Transmission Facility User's Guide
CB36	Level 6/Level 66 File Transmission Facility User's Guide
CB37	Level 6/Series 200/2000 File Transmission Facility User's Guide
CB38	Level 6/BSC 2780/3780 File Transmission Facility User's Guide
CB39	Level 6/Level 64 (Emulator) File Transmission Facility User's Guide
CB40	IBM 2780/3780 Workstation Facility User's Guide
CB41	HASP Workstation Facility User's Guide
CB42	Level 66 Host Resident Facility User's Guide
CB43	Terminal Concentration Facility User's Guide

In addition, the following documents provide general hardware information:

Order No.	Manual Title
AS22	Honeywell Level 6 Minicomputer Handbook
AT04	Level 6 System and Peripherals Operation Manual
AT97	MLCP Programmer's Reference Manual
FQ41	Writable Control Store User's Guide

Contents

Figures

Tables

Section 1

GCOS 6 Command Concepts

The GCOS 6 command processor enables users to define and control application tasks. The processor reads commands from a sequential input file (which may be an interactive terminal or a prestored command file) and causes each requested function to be executed in a serial manner.

Functions performed by the commands include file maintenance, interjob control, intrajob control, file assignment, asynchronous task operation, and operator communication.

Batch applications are always controlled by the command processor. Concurrent online applications, while not required to use the command processor, will in most cases find its use to be advantageous.

FUNCTIONAL SUMMARY OF COMMANDS

The GCOS 6 command repertoire is divided into the following functional groups:

- Execution control
- Resource control
- File and directory control
- Program preparation
- Utilities
- Interactive operations

Table 1-1 lists the commands according to function.

TABLE 1-1. FUNCTIONAL SUMMARY OF GCOS 6 COMMANDS

Command	Command Name	Function
EXECUTION CONTROL COMMANDS: These commands cause the creation, execution, suspension, and deletion of tasks and task groups from the system.		
Abort group	ABORT_GROUP	Suspends, terminates, and deletes the task group.
Associate path	ASSOC	Associates a pathname with a logical file number (LFN).
Create group	CG	Allocates and initializes all data structures that define a task group.
Create task	CT	Creates within the task group the definition of a task.
Delete group	DG	Marks the task group as eligible to be deleted when it becomes dormant.
Delete task	DT	Deletes the indicated task.
Dissociate path	DISSOC	Removes the association between the indicated LFN and the associated pathname.
Enter group request	EGR	Activates the lead task for execution or, if the lead task is active, queues the request.
Enter task request	ETR	Activates the task for execution or, if the task is active, queues the request.
Execution command	EC	Invokes the command processor to read commands from a .EC file.
Modify external switches	MSW	Modifies external switches associated with a task group.

TABLE 1-1. (CONT.) FUNCTIONAL SUMMARY OF GCOS 6 COMMANDS

Command	Command Name	Function
New process	NEW_PROC	Suspends all tasks of this task group and restarts the lead task with the original arguments.
RBT task group invocation	RBT	Invokes the remote batch terminal task group and associates it with a logical stream.
Spawn group	SG	Creates, requests the execution of, and then deletes a task group.
Spawn task	ST	Creates, requests the execution of, and then deletes a task within the issuing task group.
Status group	STG	Supplies the status of the issuing task group.
RESOURCE CONTROL COMMANDS: These commands cause the reservation and removal of files (e.g., tape or disk files or volumes, disk directories, printers, card readers, and terminal devices).		
Get file	GET	Reserves a file.
Remove file	REMOVE	Cancels the reservation of a file.
FILE AND DIRECTORY CONTROL COMMANDS: These commands manipulate and provide access to files and directories within the file system.		
Change working directory	CWD	Changes the working directory of the task group to the indicated path.
Create directory	CD	Creates a directory referenced by the indicated pathname.
Create file	CF	Creates the indicated disk file.
Create mailbox	CMBX	Creates a mailbox to contain message queues.
Delete ACL	DA	Removes entries from the access control list (ACL) of a file or directory.
Delete CACL	DCA	Removes entries from the common access control list of a file or directory.
File out	FO	Changes a user-out file to a specified file or resets it to its default value.
List ACL	LA	Lists entries of the access control list for a file or directory.
List CACL	LCA	Lists entries of the common access control list (CACL) for a specified directory.
List names	LS	Lists entries and (optionally) their attributes within a specified directory.
List search rules	LSR	Lists currently defines search rules for the task group.
List working directory	LWD	Prints the absolute pathname of the working directory.
Modify file	MF	Modifies the attributes of the indicated file.
Release file	RL	Deletes a file from the File System and releases space allocated to it.
Rename file	RENAME	Renames a directory entry with a name unique within that directory.
Set ACL	SA	Adds new entries or changes access mode of existing entries in a given access control list (ACL).
Set CACL	SCA	Adds new entries or changes access mode of existing entries in a given common access control list (CACL).
Set terminal characteristics	STTY	Changes the file characteristics of a terminal.
Walk Subtree	WS	Executes a command line in a given directory and in all subordinate directories. Prints pathname of every directory referenced on error_out.

TABLE 1-1. (CONT.) FUNCTIONAL SUMMARY OF GCOS 6 COMMANDS

Command	Command Name	Function
PROGRAM PREPARATION COMMANDS: These commands allow the user to create and modify source programs, and to create a bound unit from one or more resulting object files.		
Assembler	ASSEM	Assembles the specified assembly language program.
COBOL (entry-level)	COBOL	Compiles the specified entry-level COBOL source program.
COBOL (intermediate)	COBOLI	Compiles the specified intermediate COBOL source program.
Editor	ED	Allows the user to create or modify any text file.
FORTRAN	FORTRAN	Compiles the specified FORTRAN source program.
Linker	LINKER	Processes one or more object files to create a bound unit.
Macro Preprocessor	MACROP	Expands assembly language macro calls and %INCLUDE statements into assembly language source statements.
Report Generator	RPG	Compiles the specified Report Generator program.
UTILITY COMMANDS: These commands allow the user to perform various functions on records, files, volumes, and directories (including creating volumes, copying, comparing, dumping, and sorting).		
Compare	CPA	Compares the contents of one file or volume with another file or volume.
Copy	CP	Copies a file or volume and its file system attributes.
Copy Data Exchange	CPDE	Copies IBM files to Honeywell files or vice versa.
Create volume	CV	Creates or modifies a volume to the GCOS 6 file system standard.
Create Volume for Data Exchange	CVDE	Sets up an unformatted diskette to be acceptable on IBM equipment.
DPRINT	DP	Queues a request for deferred printing.
Edit system dump[a]	DPEDIT	Transfers to the user-out file the contents of a previously written memory dump file, or contents of current memory.
Export PAM file[a]	EX_PAM	Copies one or more sequential files to a BES1 or BES2 partitioned file.
File change[a]	FC	Changes the contents of a disk sector or control interval.
File dump	FD	Prints selected logical records within a specified file.
Import PAM file[a]	IM_PAM	Transfers one or more BES1 or BES2 partitioned file members to the file system.
ISL configurator[a]	ISL	Generates a loader to load Intersystem Link (ISL) address maps and masks.
List creation date	LCD	Lists the creation dates of a file or files in a directory.
List Data Exchange	LSDE	Lists by file name the contents of an IBM diskette.
Merge	MERGE	Merges the records of one or more sequential files.
Patch	PATCH	Modifies an object or image text file.
Print	PR	Prints an indicated file.
Reset map[a]	RS	Reconstructs the volume bit map and lists the number of unused sectors available for allocation on a disk volume.
Restore	RESTORE	Restores files previously saved by the SAVE command.

TABLE 1-1. (CONT.) FUNCTIONAL SUMMARY OF GCOS 6 COMMANDS

Command	Command Name	Function
Save	SAVE	Saves specified disk directories or files.
Set Autodial telephone number[a]	SDL	Specifies a telephone number to be used by the Autodial facility when dialing a terminal to place a line in operation.
Sort	SORT	Sorts the records on a sequential file.
Tape Position	TPOS	Manipulates the position of a magnetic tape.
File Transmission	TRAN	Transmits files between a Level 6 system and a Level 66 system.
	TRANB	Transmits files between a Level 6 system and a non-Honeywell system capable of using the IBM 2780 protocol.
	TRANH	Transmits files between a Level 6 system and a Level 6, 62, 64 or Series 200/2000 system.
INTERACTIVE COMMANDS: These commands allow the user to establish and terminate access to the system, request the execution of a batch task group, send messages to the system operator, and display the current time.		
Enter batch request	EBR	Places a request for the command processor executing in the batch task group.
Login	LOGIN	Allows a user to gain access to the system.
Message[a]	MSG	Allows a user to send messages to the system operator.
Ready message off	RDF	Suppresses printing of the ready message.
Ready message on	RDN	Restores printing of the ready message.
Terminate group request	BYE	Terminates the user's session and releases any resources reserved for him.
Time	TIME	Displays the current time.

[a] This command is made available for use only with the Mod 400 Operating System.

The remainder of this section describes certain aspects of the GCOS 6 operating system, an understanding of which is necessary to effectively use the GCOS 6 commands. The manual *System Concepts* describes the system in greater detail and should be referred to for a more complete understanding of system terms, structures, and components.

COMMAND LINE FORMAT

Commands are read and interpreted by the command processor, which executes as the lead task in the batch task group, or can execute as the lead task in an online task group. Each command causes a task to be spawned within this task group to perform the requested function (e.g., create a task within an existing group, enter a group request, dump a file). When the execution of a command terminates, control is returned to the command processor, which can then accept another command.

A command line to the processor is a string of up to 127 ASCII characters in the form:

command-name [argl . . . argn] [;command-name [argl . . . argn]] . . .

where command-name is the path name of the bound unit that performs the command's function. Each subsequent arg entry is an argument whose functions are described in following sections. The user can enter multiple commands in the same line by separating the individual commands with a semicolon.

A single command line (127 characters) can be entered on more than one physical line. To specify that command input is to be continued on a subsequent line, the user must enter an ampersand (&) as the last character in the input line. The first character of a continued

command line may not be a blank. There is no limit on the number of physical lines that can be used to enter one command line; the total number of characters in the command line, however, cannot exceed 127. A physical line consisting solely of an ampersand (&) signals the processor that a current command line is to be ignored.

ARGUMENT

An argument of a command is an individual item of data passed to the task of the named command. Some commands require no arguments; others accept one or more as indicated in the syntax of each command description. Optional arguments are enclosed in brackets; e.g., [path]. There are positional and keyword arguments (see below).

Other types of arguments are the additional arguments that follow the -ARG keyword, available in some commands, and those following path in the EC command. They represent data that is to be used in the task group being activated and are discussed in Appendix A.

POSITIONAL ARGUMENT

A positional argument is an argument whose position in the line indicates to which variable the item of data is applied. It can occur in a command line immediately after the command name or as the last argument following the control arguments, as in the LIST names command.

KEYWORD ARGUMENT

A keyword argument is a fixed-form character string preceded by a hyphen, thus -ECL. It can be alone, as in -WAIT, or it can be followed by a value, as in -FROM xx.

CONTROL ARGUMENT

A control argument is an additional argument or keyword argument whose value specifies a command option; e.g., the pathname of an alternate input or output file. In the command syntax descriptions in this manual, control arguments are denoted by the term "ctl_arg"; the argument descriptions define the specific keywords for that command. Unless otherwise noted, a control argument is optional, as indicated by enclosing brackets; i.e., [ctl_arg]. A required control argument is so described in the syntax definition, without enclosing brackets.

Except when the last argument of a command line is a positional argument, keywords of control arguments can be entered in any order in the line, following the initial positional arguments.

SPACES IN COMMAND LINES

Arguments in command lines are separated from each other by spaces. Unless otherwise indicated, a space in a command line syntax represents one or more space characters, or one or more horizontal tab characters, or a combination of these. Spaces can be embedded within an argument by enclosing the argument in single (') or double (") quote characters. If the enclosing character is also required *within* the argument, it is represented by two successive characters, thus: "NAME=""SMITH"" AREA 203".

PARAMETERS

Arguments are the user-selected items of data passed to a task. In the activated task, which is written in a generalized manner to handle any set of data passed to it, this data is known as parameters. If the activated task expects positional parameters, the order of the parameters passed to it must be in the same order as the task's positional parameters.

FILE SYSTEM PATHNAMES

The GCOS 6 File System is a tree-structured hierarchy through which each volume of storage is identified to the system. The basic element of this structure is the file. A special file, called a directory, contains information about other files.

DEFINITION OF A FILE

A file is any unit of storage outside the central processor which can supply data to or receive data from a task. A file can be a peripheral device such as a printer, card reader, or terminal, or it can be a collection of data stored within a directory structure on a magnetic (tape or disk) storage device. A source unit, object unit, listing, or bound unit is stored as a source unit file, object unit file, list file, or bound unit file, respectively.

DEFINITION OF A DIRECTORY

A directory is a file that contains information about other "subordinate" storage system entries, which in turn may represent other directories or data files. An entry named in a directory is subordinate to that directory and is "contained" within it. The information in the containing directory describes physical and logical attributes of the subordinate files.

The directory at the base of a tree structure is the *root directory*. Its name is the same as the name (volume id) of the volume where it resides.

When first created, a volume has only a root directory, within which names and attributes of subordinate directories can be created later.

All references to directories and files begin either explicitly or implicitly with a root directory name.

DIRECTORY OR FILE NAME CONSTRUCTION

A directory or file name can consist of the following ASCII characters:

- Uppercase letters (A through Z)
- Decimal digits (0 through 9)
- Underscore character (_)
- Period (.)
- Dollar sign ($)

Any name must begin with a letter or the dollar sign ($). The underscore is used to join two or more words that the system is to interpret as one name; e.g., DATE_TIME. The period separates a name from its alphabetic or numeric suffix characters. For example, in the name of a COBOL source file called COBPROG.C, COBPROG is any user specified name, and C is the required suffix, indicating to the system that this is a COBOL source file.

The length of a *root directory* name or volume identifier can be from one to six (nonblank) characters. A directory (other than the root) or a file name can have from one to twelve (nonblank) characters. A specified file name must provide for any possible suffix that might be appended by the system so that the overall length of the name does not exceed twelve characters.

PATHNAME CONSTRUCTION

A pathname is a string comprising one or more directory names and possibly one file name. All subordinate names of directories or files within a directory must be unique. The pathname describes the access path to the entity to be acted on. A pathname begins with a root directory name, followed by zero, one, or more directory names (and possibly a file name), in the order of their hierarchy.

The progressive relationship among pathname elements in the hierarchy is indicated by the following symbols:

- Circumflex (^) — denotes a *root* directory only; must precede the root directory name, with *no* intervening space (e.g., VOL011).
- Greater-than symbol (>) — indicates movement in the hierarchy away from the root. Connects two directory names or a directory name and a file name. Can also be the first character in a pathname, in which case it is immediately subordinate to the root directory of the system volume. Each successive symbol in the string indicates a change of one directory level; the name immediately following the symbol is at the next subordinate

level from the name immediately preceding it. Reading a pathname from left to right shows the access through the tree structure, away from the root, to the last element in the pathname. For example, if the root directory VOL011 contains the directory name DIR1, then the pathname for DIR1 is ^ VOL011>DIR1. However, if directory DIR1 in turn contains the file FILEA, then the pathname for FILEA is ^ VOL011>DIR1>FILEA. The > symbol is never followed by a space; nor is it preceded by a space *except* as the first character in a pathname.

- Less-than symbol (<) — Indicates movement in the hierarchy toward the root and a change of one level in that direction. Additional < symbols show successive level changes.

The last element in a pathname is the name of the entity that is to be acted on, and may denote either a directory name or a file name, according to the action to be performed.

Total length of any pathname, including all hierarchical symbols cannot exceed 58 characters, except that a working directory pathname cannot exceed 44 characters.

ABSOLUTE PATHNAME

An *absolute pathname* begins with a directory name preceded by a circumflex (^) or a greater-than symbol (>). With a circumflex, the pathname is a *full pathname*; with a greater-than symbol, the first element of the pathname is immediately subordinate to the root directory of the system volume.

RELATIVE PATHNAME AND WORKING DIRECTORY

A *relative pathname* is one that does *not* begin with a circumflex or greater-than symbol. For a relative pathname that does not begin with a less-than symbol, the first (or only) name in the pathname identifies a directory or file immediately subordinate to a directory known as the *working directory*. The working directory is the user's current position in the file system hierarchy.

The simplest form of a relative pathname has only one element, which is the name of the desired entry in the working directory.

The following are examples of relative pathnames and the full pathnames they represent when the working directory pathname is

>UDD>PROJ1>USERA

and the system was initialized from the volume SYS01.

DEVICE PATHNAMES

Reference to any device is through the Symbolic Peripheral Device (SPD) directory, which is subordinate to the system root.

DEVICE FILES (OTHER THAN DISK AND TAPE)

The general form of a device file pathname is:

>SPD>dev_name

where dev_name is the symbolic name defined for the card reader, punch, printer, or terminal device during system building.

Device files are always reserved for exclusive use (i.e., the reserving task group has read and write access but other users are not allowed to share the file).

TAPE FILES

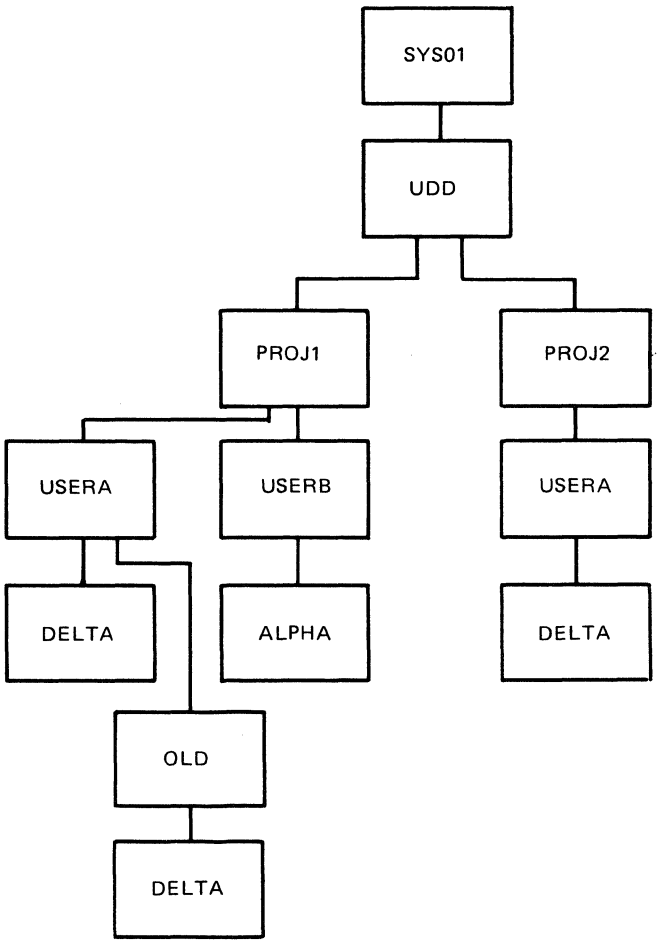
The general form of a tape file (device) pathname is:

>SPD>dev_name[>volid[>filename]]

where dev_name is the symbolic name defined for the tape device during system building, volid is the name of the tape volume, and filename is the name of the file on the volume.

Tape devices are always reserved for exclusive use (i.e., the reserving task group has read and write access but other users are not allowed to share the file).

RELATIVE PATHNAME	FULL PATHNAME
DELTA	^SYS01>UDD>PROJ1>USERA>DELTA
OLD>DELTA	^SYS01>UDD>PROJ1>USERA>OLD>DELTA
<USERB>ALPHA	^SYS01>UDD>PROJ1>USERB>ALPHA
<<PROJ2>USERA>DELTA	^SYS01>UDD>PROJ2>USERA>DELTA
<	^SYS01>UDD>PROJ1



DISK DEVICE FILES

The general form of a disk device-level access pathname is:

>SPD>dev_name[>volid]

where dev_name is the symbolic name defined for the disk device during system building and volid is the name of the disk volume.

This pathname format is used only when access to the entire volume is required (such as during a volume copy or a volume dump).

If the volid is not supplied, reservation of the disk is exclusive (i.e., the reserving task group has read and write access but other users are not allowed to share the file). This pathname form is used when creating a new volume.

If the volid is specified, reservation is read/share (i.e., the reserving task group has read access only; other users may read and write). This pathname format is used when dumping selected portions of a volume without regard to the hierarchical file system tree structure.

DEVICE PATHNAME EXAMPLES

Several examples of device pathnames are shown below.

Device	Pathname
Line printer	>SPD>LPT01
Exclusive tape volume	>SPD>MT902>VOL3
File on an exclusive tape volume	>SPD>MT902>VOL3>FILEA
Exclusive diskette	>SPD>DSK02
Nonexclusive cartridge disk volume	>SPD>RCD01>V23X

SPECIAL UTILITY PROGRAM PATHNAME CONVENTIONS

Two special pathname conventions, star name and equal name, can be used with certain utility programs to reduce significantly the number of commands required to perform a series of operations. The star name convention can be used with the COPY, COMPARE, LIST NAMES, RELEASE, and LIST CREATION DATE commands. The equal name convention can be used with the COPY and COMPARE commands. Refer to Section 2 for a full description of these commands.

STAR NAME CONVENTION

The star name convention can be used with the COPY, COMPARE, LIST NAMES, RELEASE, and LIST CREATION DATE commands to perform an operation on a group of files without having to enter separate commands for each file in the group. A star name functions in the same way as an entry name. The star name convention matches a star name with entry names in a single directory to identify a group of entries with common components. For example, if a directory contains the names of three files, PROG1.C, PROG1.O, and PROG1.L, all three files could be listed by the single command:

```
LS PROG1.*
```

instead of the three commands:

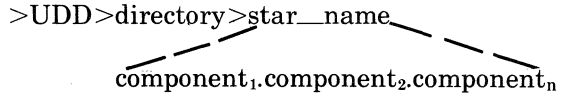
```
LS PROG1.C
LS PROG1.O
LS PROG1.L
```

The following rules apply to star names:

1. The star name convention applies only to the final entry name of a pathname.
2. The star name entry can be composed of up to 12 ASCII characters, none of which can be the less-than (<), greater-than (>), or circumflex (^) character.
3. Each star name must be made up of nonnull components, separated by periods. Thus, a star name cannot begin with a period, end with a period, or contain two or more consecutive periods.
4. When a question mark (?) character is used in a star name, the ? is treated as a special character. The ? matches any character that appears in the corresponding component and letter position of the entry name.
5. Each asterisk (*) character used in a star name matches any number of characters (including none) appearing in the corresponding component and letter positions of the entry name.
6. Only one asterisk can appear in each star name component, except that a double asterisk (double star) can appear if used as defined in rule 7.
7. A double asterisk can be used to match any number of whole components (including none) in the corresponding position of the entry name. Only one double asterisk is allowed in a star name.
8. The asterisk can be considered to represent any number of characters; the question mark can be considered to represent one character.

Two sets of examples of use follow. The first set shows the use of star names as entry name arguments. The second shows the use of star names in determining the pathname.

The following examples illustrate the use of the star name convention as the entry name argument of a command. The examples are based on the following generalized pathname:



Example 1:

LS **

Lists all entries in the working directory.

Example 2:

LS *.*.WORK

Lists all 3-component entry names, whose last component is WORK, in the working directory.

Example 3:

LS WOR?.**

Lists all entries in the working directory that have a 4-character first component whose first three characters are WOR.

Example 4:

LS *.WOR?.**

Lists all entries in the working directory that have a 4-character second component whose first three characters are WOR.

The following examples show the use of the star name convention in determining the actual pathname of a command. The examples are based on the following assumptions (D means directory, S means source file):

- The directory volid has the following entries:

Entry Name	**Type**
VOLID	D
FILE1	S
FILE2	S
DIR1	D
FILE3	S
DIR2	D
FILE4	S

- Directory DIR1 has the following entries:

Entry Name	**Type**
FILEA	S
FILEB	S
DIRA	D
FILEC	S

- Directory DIR2 has the following entries:

Entry Name	**Type**
FILEX	S
DIRX	D
DIRY	D
FILEY	S
FILEZ	S

Example 1:

LS -PN ^volid>**

Lists all entries (all is the default since no argument other than path is included) in all directories within the directory volid. This form of the LS command lists only the contents of

directories; it does not enumerate the files. The pathname ^volid>** informs the command of which directory to list. The pathname of the directory to be listed does not end with a specific entry name, but with a star name. Therefore, all directory entries within volid that conform to that star name will be listed. The listing will consist of:

```
DIRECTORY:  DIR1
        .
        . (list of DIR1)
        .
DIRECTORY:  DIR2
        .
        . (list of DIR2)
        .
```

Example 2:

```
LS  -PN  ^volid>**  -FILE
```

Lists only the files within the directories within the directory volid. The listing will consist of:

```
DIRECTORY:  DIR1
        FILEA
        FILEB
        FILEC
DIRECTORY:  DIR2
        FILEX
        FILEY
        FILEZ
```

Example 3:

```
LS  -PN  ^volid>*2
```

Lists only the directory DIR2. The listing will consist of:

```
DIRECTORY:  DIR2
        .
        . (list of DIR2)
        .
```

EQUAL NAME CONVENTION

The equal name convention can be used with the COPY and COMPARE commands to construct the output pathname entry name when the input pathname entry name has been resolved. Use of the equal name convention allows the user to employ the star convention in the input pathname and the equal name in the output pathname to copy or compare several files.

The names of the output files are constructed through the equal name convention, using either a standard input pathname entry name or an input pathname entry name that is built with the star convention.

The following rules apply to equal names:

1. An equal name is an entry name; it is composed of up to 12 ASCII characters (including spaces), none of which can be the less-than (<) or greater-than (>) character.
2. An equal name is composed of one or more nonnull components. Thus, an equal name cannot begin with a period, end with a period, or contain two or more consecutive periods.
3. When a percent character (%) appears in an equal name component, it is treated as a special character. The % character represents the character in the corresponding component and letter position of the entry name. An error occurs if the corresponding character does not exist.
4. Each equal-sign character (=) that appears in an equal name component is treated as a special character. The equal sign represents the corresponding component of the entry name identified by the star name. An error occurs if an equal sign appears in a component that contains a percent character. Only one equal sign can appear in each

equal name component, except that a double equal sign can appear if used as defined in rule 5.

5. An equal name component that consists of only a double equal sign (==) is treated as a special component. The double equal sign component represents all components of the entry names that are identified by the star name and that have no other corresponding components in the equal name. Since the double equal sign represents (corresponds to) components of the entry name identified by the star name, the equal name will have the same number of components as the entry name. Only one double equal sign can appear in an equal name.

The following examples show typical uses of the equal name convention. The examples assume two directories, DIR1 and DIR2. DIR1 is to be copied to DIR2. DIR2 is initially empty. DIR1 contains three files:

FILEA.xyz
FILEB.xyz
TESTC.xyz

Example 1:

```
CP  ^DIR1>**  ^DIR2>==
```

Copies three files from DIR1 to DIR2. DIR2 contains:

FILEA.xyz
FILEB.xyz
TESTC.xyz

Example 2:

```
CP  ^DIR1>*.xyz  ^DIR2>=.abc
```

Copies all three files, changing the second component to abc. DIR2 contains:

FILEA.abc
FILEB.abc
TESTC.abc

Example 3:

```
CP  ^DIR1>*.xyz  DIR2>TEST%.=.x
```

Copies all files. changing the first and second components and adding a third. DIR2 contains:

TESTA.xyz.x
TESTB.xyz.x
TESTC.xyz.x

The following examples have the same assumptions as examples 1 through 3, except that DIR1 is assumed to contain the following files:

A.B.C.D
X.ABC.O
WXYZ.A

Example 4:

```
CP  ^DIR1>*.*  ^DIR2>=.=
```

DIR2 contains WXYZ.A

Example 5:

```
CP  ^DIR1>*.B.**  ^DIR2>=.=.K.X
```

DIR2 contains A.B.K.X

Example 6:

```
CP  ^DIR1>*.*C.**  ^DIR2>K.X%Z.*.W
```

DIR2 contains K.XBZ.O.W

USER PROGRAM ACTIVATION

This subsection discusses two interrelated topics: the means by which the user activates his programs and the means by which he extends the system-supplied command set.

ACTIVATING A USER PROGRAM

The most direct way to activate a user program is to enter the program's bound unit pathname as the first (or only) argument in an input line (command line) to the command processor.

When the command processor reads the input line, it places the arguments of the line into a parameter block, in the order in which they appear in the line. Thus, the first entry in the parameter block is the pathname argument.

The command processor then spawns a task to load and execute the bound unit specified by the first argument in the line. When the program begins execution, register $B7 contains the address of the parameter block.

When entering the command line to activate the program, the user should be aware of the following conventions:

- *Arguments*: Only assembly language programs can handle arguments following the bound unit pathname argument.
- *Absolute and Relative Pathnames*: The first (or only) argument in the command line can be an absolute or relative pathname.

  An absolute pathname can always be used. A relative pathname can be used if the bound unit is in one of three directories searched by the loader. These directories are:

  — The working directory of the task group.

  — The system directory specified by the -LIB1 argument of the CHANGE SYSTEM DIRECTORY operator command. -LIB1 indicates the first system directory to be searched.

  — The system directory specified by the -LIB2 argument of the CHANGE SYSTEM DIRECTORY operator command. -LIB2 indicates the second system directory to be searched.
- *Entry Points*: The bound unit can have a suffix in the form ?entry, where entry is a symbolic start address within the root segment. If no suffix is given, the default start address (established when the bound unit was linked) is used.

Example 1:

```
^VOL01>TESTA>ROTA1
```

This command line uses an absolute pathname to cause the bound unit ROTA1 to be loaded and executed at its default start address.

Example 2:

```
ROTA1
```

This command line uses a relative pathname to load and execute the bound unit ROTA1 at its default start address. The working directory is ^VOL01>TESTA.

Example 3:

```
^VOL01>TESTB>CODA1?ENTRY3  1  2  3
```

This command line loads and executes the bound unit CODA1 at the symbolic start address ENTRY3. The arguments 1, 2, and 3 are placed in the parameter block after the pathname argument. CODA1 must be an assembly language program which has been written to obtain arguments (e.g., 1, 2, and 3) from the parameter block whose address is in $B7.

Example 4:

```
CODA1?ENTRY1
```

This command line causes the bound unit CODA1 (relative pathname) to be loaded and executed at the symbolic start address ENTRY1. The working directory is ^VOL01>TESTB.

EXTENDING THE COMMAND SET

The user can extend the set of commands provided with the operating system by adding commands that meet his particular requirements.

Each new command is the name of the load module used in the execution of the command.

Normally, the user places the load module into one of the system search directories (i.e., those directories defined by the -LIB1 and -LIB2 arguments of the CHANGE SYSTEM DIRECTORY command).

The procedure for activating a user program, as described above, is also the means by which the user extends the command set.

Example 1:

```
^VOL02>PROD>PAYPRT  118  315  7722
```

This user command prints payroll data for selected departments. The arguments specify that data for departments 118, 315, and 7722 is to be printed

Example 2:

```
INVENT>TT1001
INVENT>TT100A
```

These user commands list the number of 12-volt batteries (TT1001) and manual choke assemblies (TT100A) currently on hand in the users warehouse. The system directory has been defined by the -LIB1 argument of the CHANGE SYSTEM DIRECTORY command to be ZSYS51.

STANDARD I/O FILES

The following four files are always associated with the command processor:

- Command-in file
- User-in file
- User-out file
- Error-out file

The functions and characteristics of these files are described in the following paragraphs.

COMMAND-IN FILE

The command-in file for the command processor is the file from which command lines are read. Specifically, it is the device or file named by the in_path argument when a request is entered against a task group in which the command processor is executing as the lead task. The command-in file can, at times, be assigned temporarily to another device or file, as during the execution of the EC command. At the termination of execution of such a command, the command-in file reverts to the original device or file.

USER-IN FILE

The user-in file is the file from which a command function, during its execution, reads its own input. When a task group request has been processed, and as long as no alternate user-in file is specified as an argument in a subsequent command, the user-in file remains the same as the command-in file. At the termination of a command that names an alternate user-in file, the user-in file reverts to its initial assignment.

USER-OUT FILE

The user-out file is the file to which a task group normally writes its output. Certain system components (for example, compilers) also write to list files (path.L) or to the output file defined in the -COUT argument of their command. The user-out file is initially established by the

-OUT argument of the EBR, EGR, and SG commands. (Thus, originally it is the same device as the error-out file device.) The user-out file can be reassigned to another device by use of the FILE OUT command or the New User Out ($NUOUT) macro call. This reassignment remains in effect for the task group until another reassignment occurs.

ERROR-OUT FILE

The error-out file is the file to which the command processor, and any commands invoked by it, write information related to error conditions they detect. The error-out file is the same as the initial user-out file; it cannot be reassigned by a command or command argument.

FILE CONCURRENCY

The following paragraphs describe the concurrency used for standard I/O files, utility files, and program preparation files. See the GET command for a description of concurrency control.

CONCURRENCY OF STANDARD I/O FILES

Standard I/O files are reserved when a task group is spawned or requested. All nondisk standard I/O files are reserved for exclusive use. References to these files from within a task group will succeed; attempts to reserve the files from other task groups will fail. Although the operator terminal must be reserved with shared concurrency to allow read and write access by multiple groups, it can be used as a standard I/O file without any concurrency conflicts.

Disk standard I/O input files are reserved to allow multiple readers with no writers; disk standard I/O output files are reserved for exclusive use.

CONCURRENCY OF UTILITY AND PROGRAM PREPARATION FILES

Files reserved by specifying the out_path value in the -COUT argument are reserved with exclusive concurrency. Thus, multiple tasks from the same task group can write to the same output file. The output file, however, is not sharable with tasks of other task groups. For this reason, the operator terminal cannot be referenced through the -COUT argument.

CONDITIONS FOR COMMAND PROCESSOR TERMINATION

The command processor will terminate itself as the lead task of a group if any of the following occur:

- &Q is entered to the command-in file.
- End of file is encountered in the command-in file.
- An I/O error is encountered during a read from the command-in file from a noninteractive device. If the device is interactive, the system will retry the read.
- It cannot acquire user pool memory for data input from command-in buffer.
- If cannot acquire user pool memory to execute an ampersand overlay function — &P, &N, etc.
- An error is encountered in attempt to load an ampersand-related overlay.
- An error is encountered using any ampersand directive.

KEYBOARD INPUT LINE CONTROL

The terminal user has the ability to correct or delete erroneous input lines and to declare control characters to be data characters.

CORRECTING THE CURRENT LINE

To correct a character in the current line, the user presses the @ key.

Pressing the @ key deletes the previously typed character and displays an @ symbol. Each succeeding @ entry deletes another character, moving from right to left to the beginning of the line. For each deletion, the @ symbol is printed.

Examples:

RENAMR@E

Results in the line:

RENAME

RWNAME@@@@@ENAME

Results in the line:

RENAME

DELETING THE CURRENT LINE

To delete the current line, the user presses and holds the CTRL (Control) key and presses X.

Entering CTRL X deletes the current line and displays the *DEL* message on the next line; these actions are followed by a carriage return.

Example:

GOT ^ BPPKS CTRL X

Results in the line:

DEL

followed by a carriage return. The user can now enter the correct line.

DECLARING A CONTROL CHARACTER A DATA CHARACTER

To declare that a control character (e.g., @, CTRL X, CR, and \) is to be accepted as a data character, the user presses the back slash (\) key before entering the character.

The back slash is interpreted by the system as an escape character.

Example:

EGR AX TEST_A -ARG >SPD>CRD00 -CT M\@R2

The last argument required by the previously specified lead task is M@R2. If the back slash had not been entered, the character M would have been deleted.

TASK INTERRUPTION (BREAK)

The terminal user can interrupt or "break" a running task in order to reenter commands, temporarily halt the task, or terminate the task. The break can be activated by pressing the BRK (Break) or INTERRUPT key, as appropriate. (See the *Operator's Guide* for the procedures necessary to interrupt a task from the operator terminal.)

BREAK FUNCTION USAGE

Typically, a break from the interactive command-in terminal can be used to interrupt:

- Any program running in a task group whose lead task is the command processor.
- Any program invoked through a $CMDLIN (process command line) macro call issued by the lead task.

The break cannot be used with a program that is designated as the lead task in a CREATE GROUP or SPAWN GROUP command. The break can be used only under the following conditions:

- When entering from an interactive command-in terminal.
- When used to interrupt a program invoked from the lead task and by a command to the command processor.

BREAK PROCEDURES

A break is effective only with an active, running task. If the command processor is inactive, waiting for imput, pressing the Break key will have no effect.

To effect a break (task interrupt) in a running task:

1. Press the Break key
2. The system then:
   a. Truncates (possibly) the current output line
   b. Suspends temporarily the active task
   c. Puts the lead task into "break mode"
   d. Issues the break prompter message **BREAK**
3. Enter a response according to one or more of the following shown in a, b, c, or d below.
   a. Enter any command. When the entered command is *other than* SR, BYE, NEW_PROC, UW, or PI (described later in this subsection), the lead task again enters break mode and issues another **BREAK** prompter message, requesting another response. This may be followed by another command, or by one of the response commands described later in this subsection.
   b. Enter one of the following break mode responses to the **BREAK** message.
      (1) SR (Start) — Resumes execution of the suspended task; i.e., acts as though the break had not been made.
      (2) BYE (Bye) — Aborts and deletes the current task group request.
      (3) NEW_PROC (New Process) — Aborts the current task group request and re-starts the task group using the same arguments as specified in the original group request.

      Any of these commands terminates the current break (i.e.; there will be no other **BREAK** message after the command is executed).
   c. Enter UW (Unwind). If the current task is a Honeywell-supplied system program shown in Table 1-2, it terminates itself and returns all its resources.

      The break responses indicated in 3a and 3b above are also usable with these programs. Note that the programs must be running in a task group whose lead task is the command processor.

      If the terminated task was invoked following a break, the lead task reenters breakmode, issues another **BREAK** prompter message, and awaits a response.

      If the terminated task did *not* follow a break, processing continues as though the task terminated normally.

      A UW command issued to any system program other than one listed in Table 1-2 results in a 0343 or 0344 error return, followed by another **BREAK** prompter message.
   d. Enter PI (Program Interrupt). Linker and Editor output is suppressed and a return is made to the directive input level.

TABLE 1-2. SYSTEM PROGRAMS SUPPORTING THE UW (UNWIND) COMMAND

Command Name	Function	Command Name	Function
ASSEM	Assembler	LCD	List Creation Date
COBOL	COBOL Compiler	LINKER	Linker[a]
CP	Copy	LS	List Names
CPA	Compare	MACROP	Macro Preprocessor
CV	Create Volume	MERGE	Merge
DP	Dump Edit	PR	Print
ED	Editor[a]	SORT	Sort
FC	File Change	STG	Status Group
FD	File Dump		

[a]Both Editor and Linker also support the PI (Program Interrupt) command.

The PI command is meaningful only to the Editor and Linker when running in a task group whose lead task is the command processor. The commands described in 3a, 3b, and 3c above are also usable with these programs.

PI suppresses output resulting from the Linker MAP directive or from the Editor P-type directives.

UNWIND AND PROGRAM INTERRUPT COMMAND CONSIDERATIONS

The unwind (UW) and program interrupt (PI) commands are effective in user application programs only when the task to be interrupted has previously been enabled for the necessary trap. The user program must include the $TRPHD and $ENTRP macro calls for the simulated trap.

EXAMPLES OF BREAK USAGE

Example 1:

The Editor is executing a print directive and, during output, the user presses the Break key, thereby stopping further output. After the **BREAK** message appears, the user responds with PI, which returns the program to directive input level. A response of UW, instead of PI, would have terminated the Editor.

Example 2:

An LS (list names) command is executing with output going to the user terminal. The user wants to change the output path to the line printer. One possible method is:

1.	Press the Break key.	
2.	System responds to **BREAK**	Lead task enters break mode
3.	Enter FO >SPD>LPT01	File out command specifying a line printer
4.	FO execution terminates; the system issues another **BREAK** message.	
5.	Enter SR (start) command.	Resume execution of the LS command.

Another possible method is:

1.	Press the Break key.	
2.	System responds with **BREAK**	Lead task enters break mode.
3.	Enter the UW command	The current LS task terminates itself.
4.	Enter FO >SPD>LPT01	File out command specifying a line printer
5.	Enter LS	Start the list names (LS) program from the beginning.

Example 3:

This example shows successive nested break functions. Though representing a continuous procedure, the example is shown in numbered sequences for clarity.

1. The first sequence includes a command to the command processor to invoke the Editor, then to read and print the file PATH1. The Break key is pressed to interrupt the output, which was found to be from the wrong file.
2. Following issuance of the **BREAK** message, the user enters LS (list names) to obtain a display of PATH2 file names. He presses the Break key again to interrupt that LS command in order to change the pathname from PATH2 to PATH3.
3. A new LS command is entered to list the files for PATH3; however, the preceding LS command (for PATH2) is not terminated, but remains suspended. The required file is found at the beginning of the listing, the rest of the PATH3 list is not needed, so the user presses the Break key to interrupt listing of PATH3.

The following command sequences are keyed to preceding numbered descriptions.

1. Enter RDN	The system will print RDY: as each command completes execution.
Enter ED	Activates the Editor
Enter R PATH1	Read the file PATH1
Enter 1,&P	Print the file PATH1
Editor issuing print lines	
Press the Break key	Causes a break in printing
System issues **BREAK**	Command processor is in break mode
2. Enter LS -PN PATH2	List the PATH2 directory
System printing the list	User determines list is for wrong directory
Press the Break key	Causes a break in the LS command for PATH2
System issues **BREAK**	Command processor is in break mode
3. Enter LS -PN PATH3 -FILE	List files in PATH3 directory
System issuing the list	User finds desired file, no more output needed
Press the Break key	Causes break in LS command for PATH3
System issues **BREAK**	Command processor is in break mode

Subsequent actions are described as separate alternatives in Example 4 below.

Example 4:

This example consisting of five discrete actions, continues from Example 3, and in particular shows the use of the UW command to terminate successively activated tasks (i.e., unwind stacked tasks). Each part of the example is a separate procedure, independent from the others, and shows an alternative method of continuing with Example 3.

1. Start again at command level.

Enter NEW_PROC	Aborts all prior tasks; the command processor is ready for input.

2. Return to the Editor directive input level.

Enter UW	LS command for PATH3 terminates itself.
System issues **BREAK**	Since the LS for PATH3 followed a break, the command processor reenters command mode.
Enter UW	LS command for PATH2 terminates itself.
System issues **BREAK**	Since the LS for PATH2 followed a break, the command processor reenters command mode.
Enter PI	Editor is ready for the next Editor input directive.
Enter next Editor directive	

3. Return to command level by terminating in turn each previously activated task.

Enter UW	LS command for PATH3 terminates itself.
System issues **BREAK**	Command processor enters break mode.
Enter UW	LS command for PATH2 terminates itself.
System issues **BREAK**	Command processor enters break mode.
Enter UW	The Editor terminates itself.
System issues RDY:	Prompter message at command level.
Enter next command	

4. Complete the printout of PATH1 file.

Enter UW	LS command for PATH3 terminates itself.
System issues **BREAK**	Command processor enters break mode.
Enter UW	LS command for PATH2 terminates itself.
System issues **BREAK**	Command processor enters break mode.
Enter SR	Restarts printing out of PATH1 from point of interrupt.

Editor issues print lines.

5. Delete current task group request

Enter BYE	Deletes all task group request structures except the lead task. Another task group request is required to activate the lead task.

Section 2

GCOS 6 Commands

This section describes the commands by which a user exercises control over the GCOS 6 operating system. For the purpose of this section, a user is defined as any person who communicates with the operating system through a peripheral device that is the input file to the command processor (e.g., a card reader, a sequential disk file, or an MDC- or MLCP-connected terminal). This device is known as a user terminal.

In general, the commands listed in this section form a common set that can be used to direct processing under either Mod 400 or Mod 600 operating system software. In certain cases, a command (or command argument) is intended for use in a specific environment. Those commands (or command arguments) that *do not* form part of the common set are accompanied by a notation indicating the specific environment in which they are intended to be used.

This section contains complete descriptions of the formats, arguments, control arguments, and functions of the commands. In cases in which the command formats contain arguments, one or more illustrative examples of command use are given.

The command descriptions are arranged in alphabetic order to facilitate references to specific commands. A summary list of the commands, grouped by functional categories, is given in Section 1.

The following symbology is used in this section:

- Square brackets [ ] indicate an optional entry.
- Braces { } enclose information from which a choice must be made.
- The character Δ indicates a space.

ABORT GROUP

Command Name: ABORT_GROUP

Suspend, terminate, and delete the indicated online task group.

FORMAT:

ABORT_GROUP [id]

ARGUMENT DESCRIPTION:

[id]

The group identification of a task group previously created by a CG command specifying the same id. If this argument is omitted, the issuing task group is aborted.

FUNCTION DESCRIPTION:

The ABORT GROUP command causes the suspension and termination of an existing online task group, whether active or dormant. It removes the data structures which define and control the execution of the task group, and returns all memory used by the group to the appropriate memory pool. Any files open during the execution of the task group are closed. Any requests pending against the group are cancelled. The action of the ABORT GROUP command is thus similar to the DELETE GROUP command, except that the latter must wait until the task group becomes dormant, while the former takes effect as soon as all outstanding imput or output orders are complete.

This command can be issued only from an online task group.

Example:

```
ABORT_GROUP  AX
```

A task group identified as AX is terminated.

ASSEMBLER

Command Name: ASSEM

Assemble the source program unit represented by the indicated file name, applying the specified options.

FORMAT:

ASSEM path [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the source unit file to be assembled. Omit the suffix.

ctl_arg
None or any number of the following control arguments may be entered, in any order:

-COUT out_path
Listing will be written to the file out_path; a suffix (.L) is *not* appended to the file name. If this argument is omitted, the listing will be written to the file path.L in the working directory.

Note:
Path is the simple pathname, excluding the suffix appended by the Assembler.

{-LAF, -SAF, -SLIC}
Addressing mode in which source unit will be assembled. -LAF designates long-address form; -SAF designates short-address form; -SLIC designates that the source unit will be able to execute in either SAF or LAF.

Default: The mode configuration in which the Assembler is executing (must be SAF or LAF).

{-LIST_ERRS, -LE}
Specifies that only those source lines containing assembly errors, together with their error codes, are to be listed.

Default: If omitted, and -NL is not specified, the complete source program is listed, including error codes, if any.

{-CROSS_REF, -XREF}
Produces a cross-reference listing, even if -NL or -LE is specified. The listing is appended to the source listing. If there is no source listing the cross-reference listing will still be produced.

{-NO_LIST, -NL}
Suppresses source listing.

Default: Source listing produced.

{-NO_OBJ, -NO}
Suppresses object text unit output.

Default: Object text unit is generated as the file path.O in the working directory.

Note:
Path is the simple pathname, excluding the suffix appended by the Assembler.

{-SIZE nn
-SZ nn }

nn designates the maximum number of 1024-word memory blocks that may be used for the Assembler's symbol table. nn must be numeric and be from 01 through 99.

Default: 1024 words (one block).

FUNCTION DESCRIPTION:

The ASSEMBLER command is used to invoke the GCOS 6 assembler component.

The path argument can assume any of the acceptable forms of a pathname; a simple name indicates that a source program unit residing in the working directory is to be assembled. Wherever it exists, it must be suffixed with .A, indicating that it is an assembly language source unit. The path argument must be given without the .A suffix; the Assembler appends the suffix prior to searching the directory for the source unit.

If the -COUT control argument is not specified, the source listing (if requested) and/or cross-reference listing (if requested) are written to a file created by the Assembler in the working directory, having a file name of the form path.L. The path portion is the last or only element in the path argument. The file can be subsequently listed on a line printer by using the PRINT utility command. If a different file is specified by using the -COUT argument, out_path is the name of the file containing the listing. The Assembler does not append a .L suffix to out_path.

The object text unit generated by the assembler is written to a file created by the assembler, whose name is of the form path.O and which is contained in the working directory.

If files of the form path.L and path.O already exist, they are overlaid by the output generated by the current assembly.

A full description of the operation and use of the Assembler is contained in the *Assembly Language* manual.

Example:

```
ASSEM  MYPROG  -SIZE  5  -COUT  >SPD>LPT01  -XREF
```

An assembly language source program, MYPROG.A, residing in the current working directory is to be assembled. The source listing and errors are to be written to the printer LPT01, and the object text unit is to be written to the file MYPROG.O in the working directory. If MYPROG.O already exists as a result of a previous assembly, it is overlaid with the new object text unit. Five 1024-word blocks of memory are to be used for symbol resolution during the assembly. A cross-reference listing is appended to the source listing on printer LPT01.

ASSOCIATE PATH

Command Name: ASSOC

Associate the specified pathname and logical file number.

FORMAT:

ASSOC lfn path

ARGUMENT DESCRIPTION:

lfn

The logical file number by which a task is to refer to a file.

path

The pathname of the file to which the task is to refer.

FUNCTION DESCRIPTION:

The ASSOCIATE PATH command permits a task group to refer to files by the use of a standard interface known as a logical file number (LFN). The LFN serves as a "bridge" across which an input or output statement in a user program can gain access to an external file without the need to know its full pathname. This command corresponds to the Monitor macro call $ASFIL.

Conventions by which user files are identified and referred to in source programs are dependent upon the language processor by which the source program is compiled or assembled. Each processor relates an internal file identification by one means or another to a number (the LFN) which can be used in an ASSOC command to equate the internal file identification to an external pathname.

The task group within which an ASSOC command is to be issued must have been created specifying (or defaulting to) an LFN argument value large enough to include the highest LFN which is expected to be given in any ASSOC command issued during the life of the task group. This requires a knowledge of what programs are to be executed within the group and the numerical LFN values which these programs have generated.

The path argument can specify a simple, relative or absolute pathname. If a simple name is specified, the file is assumed to reside in the user's working directory. The pathname is then expanded to include the user's working directory. If, for example, the user's working directory is ^SYS01>USERA and the path argument is OLD>DELA, the expanded pathname, ^SYS01>USERA>OLD>DELA, is saved. No check is made at the time the ASSOC command is issued as to whether a file exists or note.

An incomplete pathname (e.g., OLD>) can also be associated with the LFN. With the above user's working directory, the pathname is expanded to ^SYS01>USERA>OLD. The pathname will be completed when a CREATE FILE or GET command is issued using the colon (:) option in the path argument. See Appendix A for information on the pathname colon option.

Example:

ASSOC 12 MYFILE

A file defined in a user program has been assigned a logical file number 12 by the language processor that compiled the program (e.g., the COBOL Compiler). A file, MYFILE, exists in the issuing task group's working directory. The ASSOC command relates the LFN (12), by which the program's input and output statements refer to the user file, to the external file whose pathname is ^VOL01>USERA>MYFILE.

Note:

In COBOL, the symbolic name by which the file is identified and referred to in the program (e.g., INPUT_DATA) bears no relationship to the name by which it is referred to by the File System.

BYE (TERMINATE CURRENT GROUP REQUEST)

Command Name: BYE

Terminate the execution of the current request in the issuing task group.

FORMAT:

BYE

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The BYE command causes the cessation of execution of the issuing task group. It removes all group requests defining and controlling data structures except those associated with the lead task, and returns all associated memory to the task group's memory pool. Any files that are open and in use by this task group are closed.

If the user has gained access to the system through the login procedure, typing BYE causes the message LOGOUT to be displayed at the user's terminal.

If the group was spawned or if there are no pending group requests and the group is marked for deletion, the group structures are deleted. If there is another queued group request it is executed.

CHANGE WORKING DIRECTORY

CHANGE WORKING DIRECTORY

Command Name: CWD

Change the working directory to the specified path.

FORMAT

CWD path

ARGUMENT DESCRIPTION:

path

The pathname of the new working directory. It may be a relative name or a full pathname, but cannot exceed 44 characters.

FUNCTION DESCRIPTION:

The CHANGE WORKING DIRECTORY command enables the user to move his point of reference to some other directory level within his own project's directory or to some specified point within an entirely different directory. Moving the reference point in a directory enables a task to refer, using simple names, to entities in the directory at levels other than the level established when the task was activated initially, or to entities which exist in some other directory.

If a relative pathname is given as an argument, the effect is to change the reference point within the current directory hierarchy. That is, if a user issued the command CWD MANUALS, there is assumed to exist a directory pathname within the hierarchy being used by this task. After the CWD command is executed, files that exist within the MANUALS subdirectory can be referred to by the task using simple file names.

It is also possible to traverse the hierarchy in the opposite direction, that is, in a direction toward the root. This is done by specifying as the argument the character < (less than sign) preceding the pathname. Thus it is possible to revert to the original directory level after having issued the CWD command described above by issuing a second command, CWD <. Each occurrence of the < sign moves the point of reference one level up (toward the root).

If an absolute pathname (one that begins with the > or ^ sign) is given as an argument, the effect is to move the point of reference directly to the specified point in the named directory. This directory may or may not be the same as the one being used by the issuing task.

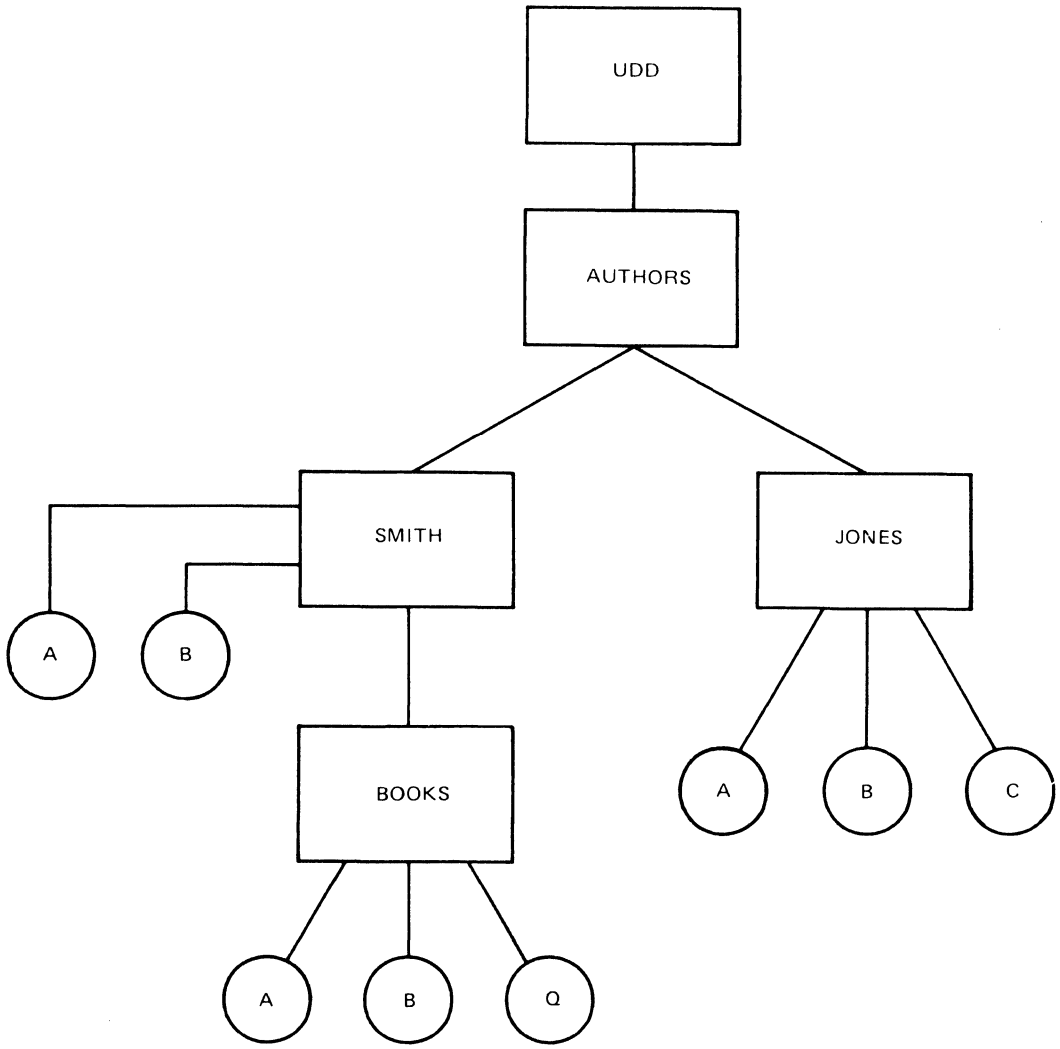
The system issues a mount message when a disk volume containing the new working directory is not mounted. The task is suspended until the volume is mounted or the operator cancels the mount request.

Example:

Assume the directory structure shown in Figure 2-1. A task group whose user id is SMITH.AUTHORS is active and is at the directory level >UDD>AUTHORS>SMITH, established when the task group was activated.

A sequence of CWD commands such as that shown below is issued. A description of the resulting action is given opposite each command.

Command	**Resulting Action**
CWD BOOKS	The point of reference is moved to the BOOKS subdirectory level (one level below the default SMITH level). Files named A, B, and Q can now be referred to by their simple names. The system supplies >UDD>AUTHORS>SMITH>BOOKS from the working directory in the construction of full pathnames for the three files.



NOTE: RECTANGLES DENOTE DIRECTORIES; CIRCLES DENOTE DATA FILES.

Figure 2-1. Typical Directory/File Structure

CWD <	The point of reference is moved up one level, back to the original SMITH level. The files named A and B in the SMITH directory (not the same files as A and B at the BOOKS subdirectory) can now be referred to by simple names.
CWD >UDD> AUTHORS>JONES or CWD <JONES	The absolute form of the pathname moves the point of reference directly to the JONES directory level. The second form achieves the same result by moving up one level to AUTHORS and then down one level to JONES.

COBOL

Command Name: COBOL

Compile the entry-level COBOL source program unit represented by the indicated file name, applying the specified compiler options.

FORMAT:

COBOL path [ctl_arg]

ARGUMENT DESCRIPTION:

path

Pathname of the source unit file to be compiled. Omit the suffix. The name must be the same as that specified in the PROGRAM ID clause of the COBOL source program.

[ctl_arg]

Control arguments; none or any number of the following control arguments may be entered, in any order:

-COUT out_path

Listing will be written to the file out_path; a suffix is *not* appended to the file name. If this argument is omitted, the listing will be written to the file path.L in the working directory. If a file other than the printer is requested, the file must already exist.

Note:

Path is the simple pathname, excluding the suffix appended by the COBOL compiler.

-DB

Compile debugging lines as comments, ignoring the WITH DEBUGGING MODE clause.

{-NO_OBJ | -NO}

Suppress object unit output.

Default: Object unit output produced as the file path.O in the working directory.

Note:

Path is the simple pathname, excluding the suffix appended by the COBOL Compiler.

{-SIZE nn | -SZ nn}

Requests nn additional 1024-word blocks of memory for complier tables. nn must be from 04 to 64. The additional memory specified in this argument is used instead of the original table size, and permits the COBOL Compiler to improve performance when compiling large programs. If you request more memory than is available, the compiler uses the available amount of memory. If specified, at least 3072 words must be available; otherwise, the compiler will use the default memory size (3000 words). If this argument is not specified, the compiler has approximately 3,000 words of memory for table space.

Note:

The following control arguments are listing options. Only one listing option may be specified at a time. Further, if no listing option is chosen, and -NL is not specified, the complete source program (along with any error codes) is listed. This is the default for all listing options shown here.

-LD

List data map, source text, errors and file map.

-LIST_ERRORS

-LE

Specifies that only the error list is to be printed.

-LIST_OBJ

-LO

List source text, data map, errors, file map and object code. This argument may not be used at the same time that -NO[OBJ] is being used.

-NO_LIST

-NL

Suppress all listings.

FUNCTION DESCRIPTION:

The COBOL command is used to invoke the GCOS 6 entry-level COBOL Compiler component. The entry-level COBOL Compiler and the object programs it generates are in short address form (SAF); neither is reentrant; and neither uses Commercial Instruction Processor (CIP) instructions.

The path argument can assume any of the acceptable forms of a pathname; a simple name indicates that a source program unit residing in the working directory is to be compiled. Wherever it exists, it must be suffixed with a .C suffix, indicating that it is a COBOL language source unit. The path argument must be given *without* the .C suffix; the compiler appends the suffix prior to searching the directory for the source unit.

If the -COUT control argument is not specified, the requested listings are written to a file created by the compiler in the working directory, having a file name of the form path.L. The path portion is the last or only element specified in the path argument. This file can be subsequently listed on a line printer by using the PRINT utility command. If a different file is specified by using the -COUT argument, the listings are written to a user-created file whose pathname is out path.

The object text unit generated by the compiler is written to a compiler-created file whose name is of the form path.O, and is contained in the working directory.

If files of the form path.L and path.O already exist in the current working directory, they are overlaid by the output generated by the current compilation.

Note: ★

The COBOL Compiler always issues a typeout, of the number of errors found, to the error-out file.

Example:

```
COBOL  CBPROG  -NO_OBJ  -LD  -COUT  >SPD>LPT01
```

A COBOL source program, CBPROG.C, is to be compiled. The source text file is located in the working directory. Listings are to include source statements, error diagnostics and a data map, and are to be written to the line printer LPT01. No object text unit is to be generated.

COBOLI

Command Name: COBOLI

Compile the intermediate-level COBOL source program unit represented by the indicated file name, applying the specified compiler options.

FORMAT:

COBOLI path [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the source unit file to be compiled. Omit the suffix. The name must be the same as that specified in the PROGRAM ID clause of the COBOL source program.

[ctl_arg]
Control arguments; none or any number of the following control arguments may be entered, in any order:

-COUT out_path
Listing will be written to the file out_path; a suffix is *not* appended to the file name. If this argument is omitted, the listing will be written to the file path.L in the working directory. If a file other than the printer is requested, the file must already exist.

Note:
Path is the simple pathname, excluding the sufflx appended by the COBOLI compiler.

-DB
Compile debugging lines as comments, ignoring the WITH DEBUGGING MODE clause.

{-NO_OBJ / -NO}
Suppress object unit output.
Default: Object unit output produced as the file path.O in the working directory.

Note:
Path is the simple pathname, excluding the suffix appended by the COBOLI Compiler.

{-SIZE nn / -SZ nn}
Requests nn additional 1024-word blocks of memory for compiler tables. nn must be from 15 to 64. The additional memory specified in this argument is used instead of the original table size, and permits the COBOLI Compiler to improve performance when compiling large programs. If you request more memory than is available, the compiler uses the available amount of memory. If specified, at least 15000 words must be available; otherwise, the compiler will use the default memory size (14000 words). If this argument is not specified, the compiler has approximately 14000 words of memory for table space.

Note:
The following control arguments are listing options. Only one listing option may be specified at a time. Further, if no listing option is chosen, and -NL is not specified, the complete source program (along with any error codes) is listed. This is the default for all listing options shown here.

-LD
List data map, source text, errors and file map.

-LIST_ERRORS

-LE

Specifies that only the error list is to be printed.

-LIST_OBJ

-LO

List source text, data map, errors, file map and object code.

-NO_LIST

-NL

Suppress all listings.

-XREF

Specifies that a cross reference listing is to be produced. A listing option other than -NL must be specified.

FUNCTION DESCRIPTION:

The COBOLI command is used to invoke the GCOS 6 intermediate-level COBOL Compiler. The compiler has the following characteristics:

- Runs in a long address form (LAF) or short address form (SAF) environment (SAF/LAF independent code (SLIC) format).
- Does not require the Commercial Processor hardware of the Commercial Simulator.
- Is not reentrant.

The object programs generated by the compiler have the following characteristics:

- Run in a LAF or SAF environment (SLIC format).
- Require either the Commercial Processor hardware or the Commercial Simulator.
- Are reentrant.

The path argument can assume any of the acceptable forms of a pathname; a simple name indicates that a source program unit residing in the working directory is to be compiled. Wherever it exists, it must be suffixed with a .C suffix, indicating that it is a COBOL language source unit. The path argument must be given *without* the .C suffix; the compiler appends the suffix prior to searching the directory for the source unit.

If the -COUT control argument is not specified, the requested listings are written to a file created by the compiler in the working directory, having a file name of the form path.L. The path portion is the last or only element specified in the path argument. This file can be subsequently listed on a line printer by using the PRINT utility command. If a different file is specified by using the -COUT argument, the listings are written to a user-created file whose pathname is out path.

The object text unit generated by the compiler is written to a compiler-created file whose name is of the form path.O, and is contained in the working directory.

If files of the form path.L and path.O already exist in the current working directory, they are overlaid by the output generated by the current compilation.

Note: ★

The COBOLI Compiler always issues a typeout, of the number of errors found, to the error output file.

Example:

```
COBOLI      CBPROG      -NO_OBJ      -LD      -COUT      >SPD>LPT01
```

A COBOL source program, CBPROG.C, is to be compiled. The source text file is located in the working directory. Listings are to include source statements, error diagnostics and a data map, and are to be written to the line printer LPT01. No object text unit is to be generated.

COMPARE

Command Name: CPA

Compare the contents of one file or volume with that of another file or volume.

FORMAT:

CPA path new [ctl_arg]

ARGUMENT DESCRIPTION:

path

Indicates the name of the file or volume to be compared. Can be any valid form of pathname; can use the star name convention (see Section 1)

new

Indicates the name of the file or volume against which that specified by the path argument is to be compared. Can be any valid form of pathname; can use the equal name convention (see Section 1).

[ctl_arg]

One or more control arguments chosen from the following list.

{-VOLUME
-VOL }

Indicates that an entire volume is to be compared, a track at a time, with another entire volume. If this argument is specified, the path and new arguments must be of the form >SPD>dev_name [>vol_id]. If vol_id is present, the volume name is verified. Inclusion of the -VOL argument means that the volid, bad track index, and the first sector of the volume directory are compared, but differences are not noted. When he specifies -VOL, the user knows that these sectors are different.

Omission of the -VOL argument means that the volid, bad track index, and the first sector of the volume directory are compared, and differences are noted. Omission of the -VOL argument also results in slower comparing of the volumes; comparing is done a sector at a time.

-CI

Indicates that a compare by control interval is to be performed. This argument can be specified under any of the following conditions:

1. The path argument represents a Series 60-compatible file and the new argument represents either a file of the same organization or a magnetic tape.
2. The path argument represents a magnetic tape containing control intervals from a type "1" COPY (refer to the COPY command description), and the new argument represents a Series 60-compatible file of the type copied to the tape.
3. Both path and new parameters represent magnetic tape files.

{-LIMIT nn
-LI nn }

Specifies that only nn records or control intervals are to be compared (if end of file is not encountered first).

{-FROM nn
-FM nn }

Specifies that the first nn-1 records or control intervals of the file are to be bypassed before beginning the compare.

{-PRINT nn | -PR nn}

Specifies that only the first nn miscompared records are to be printed. The compare operation terminates when the end of the file or volume is encountered.

In addition to the actual data printed in hexadecimal on miscompares, the address and record length of the two records is printed. For fixed relative files and all volumes, the address is the relative record number within the file. For other file organizations, the format is xxxxyy, where xxxx is the CI number and yy is the record number within the CI.

{-VERBATIM | -VBT}

This argument is available for use only when processing under Mod 400 operating system software.

If a card input file is present, it will be read in binary transcription mode. The end-of-file indicator is an 11-9-8-5 punch in column 1, followed by one or more spaces and one blank card.

FUNCTION DESCRIPTION:

The COMPARE command compares two files or volumes, record by record or control interval by control interval, and, if specified by the -PR control argument, writes the contents of any miscompared records on the user output file. If the two files or volumes are identical (do not miscompare), no response is returned to the terminal and nothing is printed. If a user is at an interactive terminal and his user output file is the terminal, he can direct the written output to another device, such as a line printer, by issuing an appropriate FILE OUT command prior to issuing the COMPARE command. At the termination of the CPA command, a message is issued to the user output file, giving the number of miscompared records or control intervals, if the number is nonzero.

If a volume compare is to be performed, the path and new parameters must represent the pathnames of peripheral devices. The dev_name portion of the pathname is the symbolic name (e.g., DSK01) given to the device in question at system building. The vol_id portion of the arguments represents the identification of the volumes to be compared.

When an entire volume is to be compared, any of the following configurations can be used:

Volume to be Compared	**Volume to be Compared Against**
Disk	Disk
Disk	Tape
Tape	Disk
Tape	Tape

The compare is executed physically; the logical organization of the volume is not considered. Comparing is done track by track.

A file compare can be performed logically or physically. A compare by control interval is a physical compare; a compare that is done by other than control interval is a logical compare.

A logical compare allows the comparing of any file organization, provided the file characteristics are otherwise compatible. For example, a file containing variable length records cannot be compared to a file containing fixed length records.

A physical compare (compare by control interval) is done in physical sequence. Control intervals are compared one at a time; logically deleted records are not considered.

If a multireel tape file has been produced (see the COPY command) and the tapes are to be compared against a disk volume, the -VOL argument cannot be specified. The compare must be done as a file compare. In the file compare, the pathname of the tape volumes must be of the form >SPD>dev_name>vol_id>filename, where dev_name and vol_id are as previously

described and filename is the name assigned by the user to the tape file when it was produced. For example:

```
CPA  >SPD>MT900>VOL00>FILEAB  SPD>RCD00>ZSYS51
```

Since no -VOL argument is present, the volume is processed as though it were a file. The HDRs on each tape volume contain a file sequence number. Therefore, the input reels must be mounted in the order in which they were created. When the end of an input reel is encountered, the system will request another input volume. As soon as that volume is mounted, processing will continue.

Note that the disk volume could also be compared against the tape volumes. The same considerations apply.

Example 1:

```
CPA  FILEA  FILEB
```

Compare two files in the working directory.

The full pathnames of FILEA and FILEB are constructed using elements of the working directory. The files are compared record by record and a summary message is issued.

Example 2:

```
FO  >SPD>LPT01
CPA  FILEA  >UDD>BOOKS>JONES>FILEA  -PR  20
```

FILEA in the working directory is compared to FILEA in the directory >UDD>BOOKS>JONES. The first 20 miscompared records are written to the line printer LPT01 along with the total number of unequal records.

COPY

Command Name: CP

Copy a file or volume.

FORMAT:

CP path [new] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the name of the file or volume to be copied. Can be any valid form of pathname; can use the star name convention (see Section 1).

[new]
Specifies the new pathname of the file or volume being copied. Can be any valid form of pathname; can use the equal name convention (see Section 1).

[ctl_arg]
- One or more control arguments chosen from the following list.

{-VOLUME
-VOL }

Indicates that an entire volume is to be copied a track at a time. If this argument is specified, the path argument must be of the form >SPD>dev_name>vol_id. The new argument must be of the form >SPD>dev_name[>vol_id] where, for dlsk volumes, vol_id cannot be included.

Inclusion of the -VOL argument means that the volid, bad track index, and the first sector of the volume directory are preserved on the output disk volume.

Omission of the -VOL argument means that the input volume is to be copied completely to the output disk volume. (In other words, the volid, bad track index, and first sector of the volume directory are copied from the input volume to the output volume.) Omission of the -VOL argument also results in slower copying of the volume; copying is done a sector at a time.

-CI
Indicates that a copy by control interval is to be performed. This argument can be specified under any of the following conditions:

1. The input is a Series 60-compatible file and the output is a file of the same type or a magnetic tape.
2. The input is a magnetic tape created by a copy under condition 1 above, and the output is a Series 60-compatible file of the same organization as that which was copied to the tape.
3. Both the path and new arguments represent magnetic tape devices. In this case the copy will be to end of volume.

Note:
In order to copy a keyed (relative, etc.) file to another keyed file without losing deleted or null records, the -CI option should be used; otherwise only active records are copied.

{-VERBATIM
-VBT }

This argument applies only to card input or output files and specifies that cards are to be read or punched in verbatim mode. The end-of-file indicator is an 11-9-8-5 punch in column 1 followed by one or more spaces, and one blank card.

FUNCTION DESCRIPTION:

The COPY command permits the creation of backup copies of files or volumes, either on magnetic tape or on a disk device. It can also be used to create copies of files in the same directory or in other directories.

The path and new arguments may express or imply the same directory portion of the file's pathname. If they do, the file name portions of both must be different. If the path and new arguments represent different directories, the file name portions of both may be the same, but the same requirement exists regarding the uniqueness of the file name in the directory represented by the new argument.

If a volume copy is to be performed, the path and new arguments must represent the pathnames of peripheral devices. The dev_name portion of the pathname is the symbolic name (e.g., DSK01) given to the device at system building. The vol_id portion of the path argument is the volume identification of the volume being copied. However, the vol_id cannot be included in the disk volume new argument. If the new argument names a magnetic tape device and the pathname includes the vol_id portion, the volume label is read and verified. The copied data then follows the volume label; i.e., the volume label is preserved. If the vol_id portion is omitted, copying begins at the current position on the tape (normally beginning of tape). In this case, the tape volume label, if any, is not preserved. A subsequent COMPARE of this unlabelled volume must be done without including a vol_ld in the pathname.

The file created by the COPY command is created just large enough to hold the data, and no larger.

When an entire volume is to be copied, any of the following configurations can be used:

Input Volume	**Output Volume**
Disk	Disk
Disk	Tape
Tape	Disk
Tape	Tape

If the output is a disk volume, that volume must have been formatted. If the output is a tape volume, a single file output volume is created. The copy is executed physically (input record equals output record); the logical organization of the input volume is not considered. Copying is done track by track.

A file copy can be performed logically or physically. A copy by control interval is a physical copy; a copy done other than by control interval is a logical copy.

A logical copy rebuilds the output file, omitting deleted records and, for indexed sequential files, also regenerates the index. A logical copy allows the copying of any file organization to any other file organization, provided the file characteristics are otherwise compatible. For example, a file containing variable length records cannot be copied to a file containing fixed length records. However, a file can be copied to tape and then copied from the tape to a disk. For example, an indexed sequential file can be copied (unloaded) to tape and the resulting sequential tape file can be copied to another indexed sequential file.

A physical copy (copy by control interval) is done in physical sequence. The first control interval on the input volume becomes the first control interval on the output volume. The copy function reads and writes one control interval at a time; logically deleted records are not recognized as such and thus are copied.

When magnetic tape volume copies are performed, the tape volumes are assumed to have the following format:

```
VOL1            (Volume label)
*               (File mark, beginning of data)
.
.
data
.
.
*               (File mark, end of data)
```

HDR and EOF labels are not maintained by the tape volume copy.

When magnetic tape file copies are performed, the tape volumes are assumed to conform to the following format:

Empty tape:

```
VOL1
HDRx
*               (File mark)  }
*               (File mark)  }  End of volume
```

Tape with data files:

```
VOL1
HDRx
.
.
*               (File mark, delimit data)
.
.
data
.
.
*               (File mark, end of data)
EOFx
.
.
*               (File mark)  }  end of recorded
*               (File mark)  }  information
```

Note that a second file written to the above tape results in the following:

```
VOL1
HDRx
.
.
*
.
.
data
.
.
*
```

EOFx
*
HDR1
.
.
*
.
.
data
.
.
*
EOFx
*
*

The double file marks indicate the end of recorded information. The file copy will load the next file into the tape by beginning its copying on the second of the two file marks.

If a disk volume is to be copied to magnetic tape and the copy will require several output tape volumes, the -VOL argument cannot be specified. The copy must be done as a file copy. In the file copy the pathname of the output volume must be of the form >SPD>dev_name>vol_id>filename, where dev_name and vol_id are as previously described and filename is the name chosen by the user to be assigned to the output tape file. For example:

CP >SPD>RCD00>ZSYS51 SPD>MT900>VOL00>FILEAB

Since no -VOL argument is present, the volume is processed as though it were a file. When a tape reel if full, the system requests that another volume be mounted. As soon as the volume is mounted, processing continues. In effect, a multireel file has been created.

Whan the tape volumes are to be copied back to disk, the same procedures are followed. The -VOL argument cannot be used; the input pathname must contain a filename. For example:

CP >SPD>MT900>VOL00>FILEAB SPD>RCD00

The output disk volume will receive its volid, bad track index, and first sector of the volume directory from the tape; none of the output disk will be preserved.

The file HDRs on each volume contain a file sequence number. Therefore, the input reels must be mounted in the same order in which they were created. When the end of an input reel is encounted, the system will request another input volume. An soon as that volume is mounted, processing will continue.

Example 1:

CP FILEA FILEB

Copy a file within the working directory. The full pathnames of FILEA and FILEB are constructed using elements of the working directory. The result of this copy is the existence of two identical files under different names.

Example 2:

CP FILEA >UDD>BOOKS>JONES>FILEA

Copy a file from the working directory to another directory on the system volume. FILEA in the working directory is copied to the directory >UDD>BOOKS>JONES, retaining the same name, FILEA, assuming that the file name does not already exist in that directory.

Example 3:

CP SUB_DIR1>FILEA VOL003>UDD>BOOKS>JONES>FILEB

Copy a file from a subdirectory in the working directory to a directory on another volume.

FILEA, one directory level below the working directory, is copied to the directory >UDD>BOOKS>JONES on a volume whose volume id is VOL003. It is assigned the name FILEB in the new directory.

Example 4:

```
CP   >SPD>DSK03>VOL001   >SPD>DSK05   -VOL
```

Copy the contents of one mass storage volume to another (like) mass storage volume. The contents of the volume VOL001, mounted on the device represented by symbolic device name DSK03, are copied to the volume mounted on the device represented by symbolic device name DSK05.

COPY DATA EXCHANGE (IBM)

Command Name: CPDE

Copy and translate an IBM file (diskette) to a HONEYWELL file or vice versa onto or from an IBM diskette; or copy one IBM volume (diskette) to another IBM volume (diskette).

FORMAT:

CPDE path1 path2 [ctl_arg]

ARGUMENT DESCRIPTION:

path1 Specifies the input path name of the file or volume (diskette) to be translated. The star name convention (see Section 1) can be used with this argument.

path2
Specifies the output path name of the file or volume (diskette) to be produced. The equal names convention can be applied to this argument.

Note:
An IBM file (data set) must be accessed through a path of the form
>SPD>DSKxx>Volid>data_set_name.

[ctl_arg]
One of the two control arguments listed below may be selected.

-VOL
Specifies that the copy is to be IBM volume (diskette) to IBM volume (diskette).

Note:
Only volume copies of diskette to diskette (IBM) are accommodated by the -VOL argument.

-TYPE x
Specifies that a file copy is to be performed, as well as the type of file copy that has been selected. The two options are:
x= 1 (IBM file to HONEYWELL file)
x= 2 (HONEYWELL file to IBM file)

Note:
The -TYPE argument is ignored if the -VOL argument is specified (i.e., for IBM to IBM volume copies).

FUNCTION DESCRIPTION:

The purpose of the CPDE utility is to transport IBM EBCDIC files to HONEYWELL files before processing under Level 6. The reverse process allows a complementary capacity for transporting Honeywell files to IBM files. The translation involves EBCDIC (IBM) to ASCII (HONEYWELL) or vice versa, and has no facility for moving packed decimal data.

CREATE DIRECTORY

Command Name: CD
Create a new directory identified by the specified pathname.
FORMAT:

CD path

ARGUMENT DESCRIPTION:

path
The pathname of the new directory to be created.

FUNCTION DESCRIPTION:

The CREATE DIRECTORY command can be used under any circumstances in which the creation of a new subdirectory within an existing directory is required. On a newly created volume, whose directory consists of only the root entry (volid) the command can be used to introduce the UDD directory level, as well as any number of project- and user-level entries (see example 4, below). On a volume which already contains user directories, this command can be used to introduce new user-level entries within a project-level directory, or new project-level entries within the UDD-level directory.

The form of the path entry of this command is the factor which determines the level of the directory being created. If it is a simple name, the name is concatenated with the entries constituting the working directory, resulting in a new directory one level below that of the working directory. A pathname consisting of more than one element results in the creation of the directory named by the last pathname element, and requires that all preceding directories named already exist (see examples 3 and 4, below).

Example 1:

CD SMITH1

Create a directory within the working directory. If the current working directory is >UDD>BOOKS>SMITH, the resulting directory is >UDD>BOOKS>SMITH>SMITH1.

Example 2:

CD <JONES

Create a new user-level directory at the same level as the working directory, and one subdirectory is >UDD>BOOKS>SMITH, the resulting new directory is >UDD>BOOKS>JONES.

Example 3:

CD <JONES
CD <JONES>JONES1

Create a new user-level directory at the same level as the worklng directory, and one subdirectory. If the working directory is >UDD>BOOKS>SMITH, the resulting directory is >UDD>BOOKS>JONES>JONES1. Note that two steps are required, since two directory levels are being created.

Example 4:

CD ^USER03>UDD

Create a new user directory of user directories on another volume which has only a volume-id, USER03. Additional project/user directories can be created on the new volume by issuing pairs of commands of the form

CD ^USER03>UDD>project
CD ^USER03>UDD>project>person

for each new directory desired. Or, if a command

CWD ^USER03>UDD

is issued first, the additional project/user directories can be created using pairs of commands of the form

```
CD   project
CD   project>person
```

CREATE FILE

Command Name: CF

Create the specified disk file.

FORMAT:

CF path [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the pathname of the file to be created. See Appendix A for the colon (:) pathname options.

[ctl_arg]
One or more control arguments chosen from the following list.

-LFN lfn
The logical file number by which a task is to refer to this file. It is a decimal value from 0 through 255. If present, the file is reserved after creation; if not present, the file is not reserved.

-F_REL
Creates a BES compatible fixed relative file without deletable records.

-N_REL
Creates a BES compatible fixed relative file with deletable records.

-SEQ
Creates a Series 60-compatible sequential file, with fixed or variable length spanned records, that is processed sequentially.

-REL
Creates a Series 60-compatible relative file, with fixed or variable length records, that can be processed sequentially or directly by relative keys.

{-INDEX / -IX}
Creates a Series 60-compatible indexed sequential record file, with fixed or variable length records, which can be processed sequentially or directly by symbolic keys.

{-CI_SIZE n / -CSZ n}
The number of bytes in a control interval for -SEQ, -REL and -INDEX type files. The value of n must be a multiple of 256 bytes. If not specified, the default is 512 bytes.

{-REC_SIZE n / -RSZ n}
The number of bytes per record for -F_REL, and -N_REL type files. For -SEQ, -REL, and -INDEX type files it specifies the maximum record size in bytes. If not specified, the default is 256 bytes.

{-SIZE n / -SZ n}
The initial size of the file in units of control intervals for -SEQ, -REL, and -INDEX type files, or in units of records for -F_REL and -N_REL type files. Default is no initial allocation.

{-INC_SIZE n / -ISZ n}
The number of units by which the file size is to be incremented whenever it must be

expanded to accommodate more data. If not specified, the value of n is the same as that specified for -SIZE. If -SIZE is not specified, n is set to 40 physical sectors.

{-MAX_SIZE n
-MSZ n}

The maximum size which this file can attain, in units of control intervals for -SEQ, -REL, and -INDEX type files, or in units of records for -F_REL and -N_REL type files. It must be set equal to the initial size, as specified by the -SIZE control argument, if a BES1-readable file is being created. If this argument is not specified, the file can expand to the physical limit of the volume.

{-KEY_OFFSET n
-KO n}

The byte offset of the first byte of the key field within the record. The first byte of a record is byte 1. This argument is required for -INDEX type files.

{-KEY_SIZE n
-KSZ n}

The number of bytes constituting the key field. This argument is required for -INDEX type files.

{-FILL_PC n
-FPC n}

The ratio of data bytes to total bytes to be put into each control interval when creating an -INDEX type file, expressed as a percentage. If not specified, the default value is 100.

-LOF nnn

For -INDEX type files nnn specifies the frequency of local overflow control intervals to be allocated when the indexed file is loaded; e.g., if nnn is 10, one local overflow control interval will be allocated after each tenth data control interval is allocated. Default is no local overflow.

{-KEY_TYPE x
-KT x}

Key component data type for -INDEX type files. Specifies the key component data type. The value of x is C for character data and D for decimal data. Default is C.

FUNCTION DESCRIPTION:

The CREATE FILE command reserves space in the file system for the specified file in accordance with the control arguments supplied in the command. It establishes a pathname whose form is dependent upon the form of the path argument and the elements of the working directory.

If a simple name is specified as the path argument, it is appended to the elements of the working directory to form the full pathname of the file. If a relative name is given, any directories expressed or implied by that relative name must exist, as must any directories expressed if the path argument is an absolute pathname.

The CF command, in effect, creates an "empty" file, which can be subsequently loaded by output statements or macro calls in user programs.

The initial shareability and permission attributes of the created file are such that the file may be referred to from both online and batch tasks, and may be read from and written to by any task. These attributes can be modified through the use of the MODIFY FILE command if different attributes (e.g., write protection) are desired.

The control arguments -F_REL, -N_REL, -SEQ, -REL, and -INDEX are mutually exclusive. If none is specified a -SEQ type file is created.

Example 1:

```
CF  FILE01  -SEQ  -CI_SIZE  1024  -SIZE  100
```

Create a file at the current level in the working directory. If the working directory is >UDD>BOOKS>JONES, the full pathname of the created file is >UDD>BOOKS>JONES>FILE01. It is a sequential file whose control interval size is 1024 bytes and whose initial size is 100 control intervals. It can be incremented in steps of 100 control intervals up to the physical limit of the volume (default values for -ISZ and -MSZ control arguments).

Example 2:

```
CF  SUB_DIR1>MYFILE  -IX  -SIZE  50  -KO  9  -KSZ  6  -MSZ  200
```

Create a file in an existing directory one level below the current level in the working directory. Given the same working directory as in the previous example, the full pathname of the created file is >UDD>BOOKS>JONES>SUB_DIR1>MYFILE. It is an indexed file whose initial size is 50 control intervals of 512 bytes, and whose increment size and maximum size are 50 and 200 control intervals, respectively. The first byte of the record key is the ninth byte of the record (the first byte of a record is byte 1), and the key is six bytes long.

The values provided with the -SIZE and/or the -INC_SIZE arguments cannot cause the extent to exceed 8191 physical sectors. The actual limits, in terms of actual supplied value, depend upon the type of device and the -CI_SIZE.

CREATE GROUP

Command Name: CG

Perform the initialization functions necessary to the initiation of an online task group.

FORMAT:

CG id base_lvl [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of the new task group. It is a 2-character name that cannot have the $ as its first character.

base_lvl
A base priority level, relative to the system level, at which all tasks in this task group will execute. A base level of 0, if specified, is the next higher level above the last system priority level. The sum of the highest system physical level plus 1, and the base level of the group, and the relative level of a task within that group must not exceed 62_{10}.

[ctl_arg]
One or more control arguments chosen from the following list.

{-EFN root
-EFN root? entry}

The name of a bound unit root segment to be loaded as the lead task if it is not already loaded and linked as sharable. The root segment name can be suffixed with?entry, where entry is a symbolic start address within the root segment. If ?entry is not given, the start address established when the bound unit was linked is assumed.

-ECL
The root segment of the command processor is to be loaded as the lead task.

-LRN n
Specifies the highest logical resource number (LRN) that will be referred to by any task in the task group. The maximum value is 252. The default value is the highest LRN used by the system.

-LFN n
Specifies the highest logical file number (LFN) used by any task in the task group. The maximum value is 255. The default value is 15. Refer to the ASSOCIATE PATH or GET FILE command.

-POOL id
The name of the memory pool from which all dynamic memory required by this task group is to be taken. id is a 2-character ASCII pool identifier; if specified, id must have been defined at system building. If this argument is not specified, the issuing task group's memory pool is used. The -POOL id argument is intended for use under the Mod 400 executive only.

Note:
-EFN or -ECL, but not both, can be specified. If neither is specified, -ECL is assumed.

FUNCTION DESCRIPTION:

The CREATE GROUP command causes the initialization and allocation of all data structures used by the system to define and control the execution of the task group. It causes the loading of the root segment of the lead task of the task group. It does *not* cause the system to activate any task within the task group.

This command can be issued only from an online task group.

Example:

```
CG  AX  5  -EFN  MAIN_PG?ENTRY1  -LRN  8  -POOL  A2
```

A task group identified as AX is created. The lead task of the group is the program MAIN_PG, whose execution is to be started at the symbolic address ENTRY1. No task in the group will execute at a relative priority level lower than 5, nor refer to a logical resource number higher than 8. Memory will be obtained from a pool identified as A2 at system building.

CREATE MAILBOX

CREATE MAILBOX

Command Name: CMBX

Create a mailbox to contain the message queues used in communicating between task groups.

FORMAT:

CMBX name [ctl_arg]

ARGUMENT DESCRIPTION:

name

Name of mailbox (up to 12 characters). Can be an absolute or simple pathname.

Note:

To use a simple pathname the user must have previously created a mailbox root directory named MDD.

[ctl_arg]

The following control arguments *must* be specified.

-MEM pool-id

Indicates that message queuing is to be performed in memory; queuing is done in the memory pool identified by pool id.

-OW

Specified that mailbox is to have one-way capability.

FUNCTION DESCRIPTION:

The CMBX Command creates a directory corresponding to the mailbox name and a file ($MBX) within that directory defining the mailbox attributes. When a task group sends a message to another task group, it sends to a named mailbox; when a task group receives a message from another task group, it receives from a named mailbox. Under Mod 400 the only queuing supported is memory queuing. It is advisable for the user to dedicate a memory pool for messages.

Before the user issues a CMBX command, he should have created a mailbox root directory to contain the simple names of the mailboxes. (If the mailbox root directory is not named MDD, then simple pathnames cannot be used in the CMBX command.)

The user should set access on the mailboxes such that the task group sending a message has list access on the directory defining the mailbox and the task group receiving the message has read access on the $MBX file for the mailbox.

Refer to the *System Service Macro Calls* manual for details on the intergroup message facility macro calls.

Example:

```
CD  >MDD
CREATE_MBX  >SMITH  -MEM  -SIZE 100
```

Create the mailbox root directory named MDD. Create a mailbox whose directory name is >MDD>SMITH and whose file name is >MDD>SMITH>$MBX. *Queuing* is to occur in memory; the queue size is 100 bytes.

CREATE TASK

Command Name: CT

Perform the initialization functions necessary to the initiation of a task within the issuing task group.

FORMAT:

CT lrn rel_lvl ctl_arg

ARGUMENT DESCRIPTION:

lrn
The logical resource number (LRN) by which the issuing task group can refer to the created task. It cannot exceed the value specified by the -LRN control argument in the CREATE GROUP command which created the group of which this task is a member.

rel_lvl
The priority level, relative to the task group's base priority level, at which the created task is to execute.

ctl_arg
One or more control arguments chosen from the following list.

{-EFN root
-EFN root?entry}

The name of the bound unit root segment to be loaded for execution. The root segment name can be suffixed with ?entry, where entry is a symbolic start address within the root segment. If no suffix is given, the default start address, established when the bound unit was linked, is assumed.

{-SHARE lrn [ssa]
-SHR lrn [ssa]}

This argument is available for use only when processing under Mod 400 operating system software.

The same bound unit in the same task group is used as for the task identified by lrn. (This task must have been previously defined by a CREATE TASK command specifying this lrn.) ssa is the symbolic start address within the root segment of the task lrn. If none is given, the root segment's default start address, established when the shared bound unit was linked, is assumed.

Note:
In any invocation of the CT command, -EFN *or* -SHARE, but not both, must be specified.

FUNCTION DESCRIPTION:

The CREATE TASK command causes the allocation and initialization of the data structures which define and control the execution of a task. It causes the loading of the root segment specified by the -EFN control argument. If does *not* activate the task (the ENTER TASK REQUEST command is required to perform activation).

One or more CT commands can be issued to create one or more tasks within the task group. These tasks can be requested for execution concurrently or serially by entering the appropriate control argument in the ETR command which is used to activate each task. Refer to the description of the ETR command.

Example:

```
CT  10  2  -EFN  PROG10
CT  11  3  -EFN  PROG11
CT  12  2  -SHARE  10  ENTRY2
```

Three tasks are made known to the issuing task group. Their logical resource numbers (LRNs) are 10, 11, and 12. Task 10 is to execute at priority level 02 relative to the base priority level established when the task group was created. Task 11 is to execute at relative level 03, and task 12 is to execute at the same relative level as task 10. If the task group's base level was resolved to 20, then the three tasks execute at physical priority levels of 22, 23, and 22, respectively. Task 12 is to share the same bound unit as task 10; however, execution of task 12 begins at a different point in the bound unit, specified by the label ENTRY2, (task 10's entry point is the default entry point established when PROG10 was linked). Subsequent ENTER TASK REQUEST commands cause the execution of the above tasks to begin (refer to the description of the ETR command).

CREATE VOLUME

Command Name: CV

Create or modify a volume.

FORMAT:

CV path ctl_arg

ARGUMENT DESCRIPTION:

path

The pathname of the device upon which the volume to be created is mounted. The form of the pathname is

>SPD>dev_name[>vol_id]

If vol_id is present, the volume name is verified.

ctl_arg

Only one control argument from the following, except that either the -DBLOC or -SIZE argument, or both, may be specified only with the -FORMAT argument.

{FORMAT vol_id [t] [nn]
-FT vol_id [t] [nn] }

Assign vol_id as the volume id and the d sk volume major directory name. For a disk volume, preformat the volume by initializing all sectors to zero, checking for bad sectors, and creating the olume id, the volume major directory, the bit map and the defective sector index.

For a storage module volume, give it logical sector size nn, where nn may be 8, 16, 32, or 64. Default is 8 for all storage module device types except 2363, for which it is 16.

The t character defines the format of a magnetic tape volume where possible values are 1, 2, 3, or H (default is 3) which specify the following formats:

1 — American National Standard Institute level 1

2 — American National Standard Institute level 2

3 — American National Standard Institute level 3

H Honeywell derivative of American National Standard Institute level 3.

If used when formatting a disk volume, the optional t character is ignored.

{-DLOC aaaa
-DL aaaa }

This argument is available for use only when processing under Mod 400 operating system software.

Causes the disk volume directory to start at sector aaaa. The value aaaa can be decimal, or a hexadecimal number X'hhhh' in which 'hhhh' represents four hexadecimal digits. This argument can be used only when -FORMAT is specified and may be used when -SIZE is specified.

{-SIZE ssss
-SZ ssss }

This argument is available for use only when processing under Mod 400 operating system software.

Causes the disk volume directory length to be established as ssss physical sectors. The value ssss can be decimal, or a hexadecimal number X'hhh' in which 'hhh' represents three hexadecimal digits. This argument can be used only when -FORMAT is specified and may be used when -DLOC is specified.

{-BOOT [X'hhhh']
-BT [X'hhhh']}

This argument is available for use only when processing under Mod 400 operating system software.

Create bootstrap records and write them to volume-relative sectors 0 through 6. The existing volume id and major directory name are not modified. The X'hhhh' field defines certain bootstrap options as described in the function description. If used, this value becomes permanent and cannot be overridden at startup.

-ISL [X'hhhh']

Create Intersystem Link (ISL) bootstrap records and write them to volume-relative sectors 0 through 6. The existing volume id and major dlrectory name are not modified. The X'hhhh' field defines certain bootstrap options as described in the function description.

{-MDUMP nn
-MD nn}

This argument is available for use only when processing under Mod 400 operating system software.

Create a memory dump bootstrap record and write it to volume-relative sector 0. The existing volume id and major directory name are not modified, nn specifies the number of 4096-word modules to be dumped.

Create a file named DUMPFILE on the volume, large enough to contain a dump of nn 4K modules of memory. Put a MDUMP record on sector 0 of the volume that will dump nn 4K modules of memory into DUMPFILE, to be printed subsequently by DPEDIT. The default value of nn is 6.

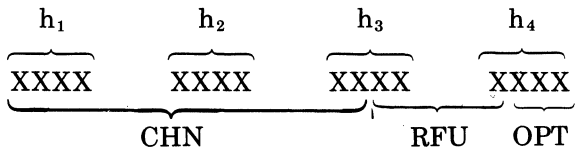
{-RENAME y
-RN y}

Change the volume id and major directory name to that specified by y. y is a one- to six-character ASCII string. A tape volume cannot be renamed.

FUNCTION DESCRIPTION:

The CREATE VOLUME command initializes a tape or disk volume in one of several ways. A previously unused disk volume can be assigned a volume identification through the use of the -FORMAT control argument. This argument, in addition to initializing all tracks on the volume and verifying their integrity, writes a volume label record containing the volume identifier specified by the vol_id field in this argument. It also establishes this identifier as the volume major (root) directory name. Thus, if vol_id is given the value USER01, the volume label contains this value as the volume identifier, and the root directory pathname for this volume is ^USER01.

A volume which has already been assigned a volume identifier as described above can be supplied with a bootstrap routine in one of two forms. The -BOOT control argument causes a standard system bootstrap routine to be written on the volume, the ISL argument causes a standard ISL bootstrap routine to be written. The X'hhhh' field of these two arguments is used to define the channel of the disk device containing the directive files and routines used during system initialization, and to define certain bootstrap-and initialization options. The field consists of four hexadecimal digits whose bit configuration is broken down as follows:



CHN

Ten bits (bits 0 through 9) which specify the channel number of the initialization device (e.g., 0400, 1280). The fourth digit of the channel number is always zero, and the values that can be assumed by the third digit are 0, 4, 8, and C (hexadecimal).

RFU

These bits (bits 10 through 12) are reserved for future use and must be zero.

OPT

These bits (bits 13 through 15) establish the bootstrap/initialization options as follows:

If bit 13 = 1: Halt at the conclusion of the system bootstrap routine, and before entering the operating system initialization code.

If bit 14 = 1: Use the Honeywell-supplied directive file on the device specified by CHN.

If bit 15 = 1: Bootstrap from the fixed cartridge disk device specified by CHN.

The -MDUMP control argument causes a special record, which is the memory dump routine, to be written on the volume. A file, DUMPFILE, is allocated with a sufficient number of sectors to contain the number of memory words specified by the xx field of the -MDUMP argument.

A volume already having a volume identifier can be given a new identifier through the use of the -RENAME control argument. This causes the volume identifier field of the volume header record, and the root directory name, to be changed to the identifier specified by the y field of this argument.

The CV command must specify the pathname of the peripheral device (cartridge disk, diskette, storage module, or tape) upon which the volume to be initialized is mounted. The dev_name portion of the path argument is the symbolic name of this device as defined at system building. The vol_id field of the path argument, if used, indicates that the volume already has a volume identifier, and that this identifier is to be checked for agreement with a specified identifier. If the two identifiers do not agree, an error message is used and the command is terminated. The vol_id field of the path argument does *not* assign an identifier or root directory name to the volume; this can only be done by using the -FORMAT control argument.

The system recognizes unique vol_ids. If disk volumes of the same vol_id are used, it is necessary to rename one of the volumes before the system accepts it. Simply follow the procedure for an unformatted volume, and invoke CV -RENAME rather than -FORMAT. Mount the volume only at the appropriate remount message and processing (i.e., the volume rename) will continue. If the CV is attempted and another volume is the same vol_id is mounted, a dismount message will be issued after the vol_id is written on the volume. If the -MDUMP option is requested, the create volume will issue a Create File macro call which will attempt to place the new file on the duplicate named volume, not on the one just created.

Only one of the parameters -BOOT, -MDUMP, -FORMAT and -RENAME may be specified at a time.

To format an unformatted diskette, wait until a "mount" message is encountered after CV has been loaded and begun execution. At this point, mount the unformatted pack and processing will continue.

To format an unformatted removable cartridge disk or storage module, keep the disk in the "off" condition until the "Mount" message is received after CV has been loaded and executed. At this point, cycle-up the disk and processing will continue.

To format an unformatted fixed cartridge disk, proceed in the same way as for the removable disk except when the fixed disk is on the same channel as the executing removable system pack. For the latter case, at the mount message simply cycle down and up the cartridge unit and processing will continue.

To set up a volume label on a magnetic tape, use the following command form:

CV>SPD>sympd [>vol_id1] -FT vol_id2 x

where "x" is a hexadecimal character which may be 1, 2, 3 or H and which is used as the last character of the tape header record. The default for this is 3. If the optional character is used when a disk is formatted, it is ignored.

Ignore the error message 020107(26 cccc 0100 000) encountered when using unformatted volumes.

Example 1:

```
CV  >SPD>DKS03  -FT  USRDTA
```

A volume mounted on the device identified at system building as DSK03 is to be formatted and assigned the identifier and root directory name USRDTA. If this volume is to contain only user data (i.e., it is not to be used for system initiation or dumping of memory), no further initialization is required. That is, no bootstrap records need be created for this volume. Other directories can be established under the root directory USRDTA by subsequent use of the CREATE DIRECTORY command.

Example 2:

```
CV  >SPD>DSK02  -FT  DMPVOL
CV  >SPD>DSK02>DMPVOL  -MD  04
```

A volume mounted on the device identified as DSK02 is to be formatted and assigned the identifier and root directory name DMPVOL. This volume is to be used for dumping memory, and is therefore (by the second CV command) given a memory dump bootstrap record. Dumps are to contain four 4096-word modules of memory. The second command also specifies that the previously assigned volume identifier is to be verified prior to creation of the memory dump bootstrap record.

Use of the optional volume id as part of the pathname insures that the proper volume is mounted, thus avoiding concurrency errors.

Finally, this utility automatically reserves the innermost cylinder on every disk pack (except diskette) for T&V usage.

CREATE VOLUME DATA EXCHANGE (IBM)

Command Name: CVDE

Create a volume (diskette) for data exchange which will be acceptable on IBM equipment.

FORMAT:

CVDE path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The path name of the device upon which the volume (diskette) to be created is mounted. The form of the path name is

>SPD>sympd [>vol-id]

[ctl_arg]
The only control argument accompanying the CVDE command is as follows:

-FT vol_id
Specifies the volume id being assigned to the volume (diskette) being created.

FUNCTION DESCRIPTION:

The purpose of the CVDE command is to set an unformatted volume (diskette) to a 3740-like format to make it acceptable on IBM equipment.

DEFERRED PRINT

Command Name: DP

Queue a request for deferred printing for the indicated file.

FORMAT:

DP path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the file whose contents are to be printed

[clt_arg]
One or more control arguments chosen from the following list:

{-LIMIT nn
-LI nn}

Specifies the number of records to be printed if end of file is not encountered before the value of nn is satisfied. If not specifiied, all records in the file are printed.

{-COPIES n
-CPn}

Specifies the number of copies to be printed, i.e., the number of times the file is to be printed for this invocation. Default is 1.

{-SPACE n
-SP n}

This argument indicatyes that the file is not a true print file with print control characters in its records. Each record is printed on one or more print lines. The value of n specifies the line spacing between records, and can be either 1 or 2. 1 specifies single spacing (no blank line). 2 specifies double spacing (one blank line). The default value for n is 1. If this parameter is not specified, the first record byte is treated as a printer control character, i.e.; the file is assumed to be a print file. See the control byte description for the printer driver in the *System Service Macro Calls* manual.

{-FORTRAN
-FTN}

The print file was created by a FORTRAN object program and has print control characters of the FORTRAN type.

{-FROM nn
-FM nn}

Indicates that the first nn records of the file are to be skipped before printing begins. If not specified, printing starts at beginning of file.

{-LINE LEN nn
-LL nn}

Specifies the number of characters to be printed per line. If a longer line is read from the file, it is folded at the indicated print position. If not specified, the value of nn is 68.

{-RELEASE
-RL}

Specifies that, at the completion of printing, the file is to be released.

-DESTINATION string
-DS string

Use the value of the specified "string" for the destination field of the printing heading sheet. For spaces to be included in the destination field, the supplied character string must be enclosed in quotes. This field can be up to 13 characters long. If the -DS argument is omitted, the person id is printed.

-HEADING string
-HE string

Use the value of the specified "string" for the heading field of the printing heading sheet. For spaces to be included in the heading field, the supplied character string must be enclosed in quotes. This field can be up 26 characters long. If the -HE argument is omitted, the account id is printed.

FUNCTION DESCRIPTION:

DPRINT verifies the path name and control arguments and then enters a request for a deferred file print to $P. After the print request has been submitted, the user is allowed to log off without losing the print.

DELETE ACCESS CONTROL LIST

Command Name {DA | DELETE_ACL}

Delete entries from the ACL of a file or directory.
FORMAT:

{DA | DELETE_ACL} [path user_id] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the pathname of a file or directory. If this argument is omitted or if -WD is entered, the working directory is specified. If it is omitted, user_id cannot be specified.

user_id
Specifies an access control name that must be of the form person account mode. All ACL entries with matching names are deleted. (For a description of the matching strategy, refer to the SET_ACL command.) If path is specified, user_id should also be specified. If user_id is omitted, the system_id of operator. system.* is used.

[ctl_arg]
One or more control arguments from the following list.

{-A | -ALL}

Causes all ACL entries to be deleted. This argument overrides user_id, if both are specified.

{-BF | -BRIEF}

Suppresses the message "USER NAME NOT ON ACL"

FUNCTION DESCRIPTION:

This command removes entries from the access control list (ACL) of a file or directory. The user must have modify access to the containing directory in order to delete entries.

If the command is invoked with no arguments, it deletes the entry for the user's person. account.* on the ACL of the working directory.

DELETE COMMON ACCESS CONTROL LIST

Command Name: {DCA | DELETE_CACL}

Delete entries from the CACL of a directory.

FORMAT:

{DCA | DELETE_CACL} [path user_id] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the pathname of a directory. If this argument is omitted or if -WD is entered, the working directory is specified. If it is omitted, user_id cannot be specified.

user_id
Specifies an access control name that must be of the form person.account.mode. All CACL entries with matching names are deleted. (For a description of the matching strategy, refer to the SET_ACL command.) If path is specified, user_id should also be specified. If user_id is omitted, the system_id of operator. system.* is used.

[ctl_arg]
One or more control arguments from the following list.

{-A | -ALL}

Causes all CACL entries to be deleted. This argument overrides user_id, if both are specified.

{-DIR | -DIRECTORY}

Causes directory CACL entries to be deleted

-FILE

Causes file CACL entries to be deleted

{-BF | -BRIEF}

Suppresses the message "USER NAME NOT ON CACL"

FUNCTION DESCRIPTION:

This command removes entries from the common access control list (CACL) of a directory. The user must have modify access to the containing directory in order to delete entries.

If the command is invoked with no arguments, it deletes the entry for the user's person. account.* on the file CACL of the working directory. If -DIR and -FILE are both specified, both directory and file CACL entries are deleted. If neither is specified, only file CACL entries are deleted.

DELETE GROUP

DELETE GROUP

Command Name: DG

Mark the online task group as eligible for deletion when it becomes dormant.

FORMAT:

DG id

ARGUMENT DESCRIPTION:

id

The group identification of a task group previously created by a CG command specifying the same id. The default is to delete the issuing task group.

FUNCTION DESCRIPTION:

The DELETE GROUP command removes all data structures constructed by the CG command issued previously with this id. No more ENTER GROUP REQUEST commands can be issued for this task group after the DG command has been executed. The DG command takes effect immediately if the task group is dormant when the command is issued. If it is active (i.e., if its code is being executed and/or there are still requests in this task group's request queue), the DG command takes effect when execution terminates and there are no more requests in the queue.

When a task group is deleted, the memory occupied by the data structures defining the group, and any memory associated with the execution of the group, is returned to the appropriate memory pool and is available for use by other task groups.

This command can be issued only from an online task group.

DELETE TASK

Command Name: DT

Mark the online task as eligible for deletion of its definition from the task group when the task becomes dormant.

FORMAT:

DT lrn

ARGUMENT DESCRIPTION:

lrn
The logical resource number of the task to be deleted.

FUNCTION DESCRIPTION:

The DELETE TASK command removes from the task group all of this task's defining and controlling data structures, and returns this task's memory to the task group's memory pool. If the task is currently dormant, it is immediately deleted from its task group. If it is not dormant, the task is deleted when it terminates and no more task requests are queued against it. After this command is issued, no more task requests (ETR) will be accepted for this task without again creating the task.

Example:

DT 12

The task whose logical resource number is 12 is to be deleted.

DISSOCIATE PATH

Command Name: DISSOC

Break the association between the indicated logical file number and the external file name (pathname), established by a previous ASSOCIATE PATH command.

FORMAT:

DISSOC lfn

ARGUMENT DESCRIPTION:

lfn
The logical file number whose association with an external file name is to be broken.

FUNCTION DESCRIPTION:

The DISSOCIATE PATH command is used when a task has no further need for the association between the specified logical file number (LFN) and the related external file name. It frees the LFN so that another task, which requires the same LFN to be associated with a different external file name, can have this relation made by the use of another ASSOC command.

The DISSOC command has no effect on a file which is reserved or open at the time the command is issued; i.e., it does not dissociate the file from the LFN. (This is done by a REMOVE command.) It merely breaks the connection between a name and the LFN which was established by ASSOCIATE PATH.

Example:

DISSOC 12

The external file name associated with LFN 12 is broken. Existing connections between the LFN and file name at the time of the dissociate action continue until the file is removed.

DUMP EDIT
[MOD 400 ONLY]

DUMP EDIT

Command Name: DPEDIT

Transfer to the user-out file the contents of a previously written memory dump file. The user-out file must be a device that provides 132 print positions.

FORMAT:

DPEDIT [path] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the memory dump file to be printed.

ctl_arg
Zero, one, or more of the following control arguments may be entered, in any order:

{-NO_LOGICAL / -NL}
No logical dump of system control structures produced.
Default: Logical dump produced.

{-NO_PHYSICAL / -NP}
No physical dump of memory produced.
Default: Physical dump produced.

{-GROUP id [id] . . . [id] / -GP id [id] . . . [id]}
Produces only group-related information within a logical dump for the group(s) indicated by id; id is the 2-character group identifier.

{-FROM X'address' / -FM X'address'}
Low-memory address of area that will appear in physical dump; must be a physical address specified in hexadecmal.
Default: Absolute 0.

-TO X'address'
High-memory address of area that will appear in physical dump; must be a physical address specified in hexadecimal.
Default: Physical high memory address of the dump file.

{-MEMORY / -MEM}
Produces a dump of main memory. If both the path argument and this argument are specified, the path argument is ignored.
Default: A dump is produced of the file specified in the path argument.

Note:
Either the path argument or the -MEMORY control argument must be specified.

FUNCTION DESCRIPTION:

The DUMP EDIT command causes the transfer to the user-out file of the contents of the memory dump file obtained through the MDUMP utility described in the *Program Execution and Checkout* manual. This file is allocated at the time a volume is created with a memory dump bootstrap record, and comprises sufficient sectors to contain the number of 4096-word memory modules specified when the volume was created. Refer to example 2 in the description of the CREATE VOLUME command.

The transfer of data from memory to the memory dump file must have been previously performed by a bootstrap operation which specified the channel to which the device containing the memory dump volume was attached.

The memory dump output produced by the DPEDIT command comprises a logical portion and a physical portion. The former consists of an edited printout of system and user control structures such as a system summary, memory pool data, file system structures, task control blocks, dedicated memory locations, and group control blocks. The physical portion consists of a memory image printout encompassing the memory locations explicitly or implicitly specified by the -FROM and -TO arguments or their defaults. This portion of the dump is printed in both hexadecimal and ASCII representation, with duplicate line suppression. Any hexadecimal digit pairs which have no printable ASCII equivalents are represented by ASCII period (.) characters.

Additional information concerning the Dump Edit utility program is contained in the *Program Execution and Checkout* manual.

Example 1:

```
DPEDIT  ^DUMPER>DUMPFILE  -FROM  X'0400'  -NL
```

Print the contents of the memory dump file, DUMPFILE, located on the volume DUMPER. Memory locations between 0400_{16} and the end of the dump file are printed. The logical portion of the dump is to be omitted. The memory dump is printed on whatever device is currently serving as the user-out device (a wide-carriage, 132 characters per line, user terminal if no FILE OUT commands have been previously issued).

Example 2:

```
FO  >SPD>LPT01
DPEDIT  ^DUMPER>DUMPFILE
```

Print the contents of the same dump file as in example 1 above. In this case, because of the FO command which precedes the DPEDIT command, the output is printed on a printer designated as LPT01. Both logical and physical portions of the dump are printed, and the physical dump encompasses locations zero through the end of the dump file.

EDITOR

Command Name: ED

Add, delete, or modify selected lines of a source unit file.

FORMAT

ED [ctl_arg]

ARGUMENT DESCRIPTION:

[ctl_arg]
One or more control arguments chosen from the following list.

-IN path
Specifies the file from which Editor directives are to be read and to which Editor output is to be directed. If not specified, directives are obtalned from the current user-in file and output is sent to the task group's user-out file (as specified during the group request).

{-LINE_LEN n
-LL n}
Specifies the maximum line length to be acted upon by the Editor. If not specified, n assumes the value of 80.

{PROMPT
-PT}
Stipulates that the prompt characters "E?" (no carriage return) are to be printed to the user_in file upon completion of the previous Editor directive. If the user_in is other than a terminal-like device, the -PT argument is ignored.

-NBS
Specifies "no blank suppression." This argument stipulates that the Editor does not suppress trailing blanks on the input line.

FUNCTION DESCRIPTION:

The EDITOR command is used to invoke the Editor component. Execution of the Editor is normally intended for the batch task group, as is the execution of most of the other components used in a program development activity.

A full description of the operation and use of the Editor is contained in the *Program Preparation* manual.

ENTER BATCH REQUEST

Command Name: EBR

Enter into the batch request queue a request for the execution of the command processor.

FORMAT:

EBR in_path [ctl_arg]

ARGUMENT DESCRIPTION:

in_path
Name of the file from which the command processor is to read its commands.

[ctl_arg]
One or more of the following arguments can be entered.

-OUT out_path
Defines the pathname of the file to receive user output (user-out file) and error output (error-out file) from the batch task group. If this argument is not specified, one of the following assumptions is made:

- If in_path specifies a disk file, out_path is in_path.AO
- If in_path specifies an interactive terminal, out_path is in_path
- If in_path specifies an input-only device, out_path is null

-WD path
Specifies that path is to be used as the working directory. This argument is set to null if not specified

-ARG arg arg . . . arg
Indicates that additional arguments required by the task group during execution follow. These additional arguments are passed to the command processor to be used as necessary, and are substituted for parameters in the command-in file. If used, the -ARG control argument must appear last. Refer to Appendix A for an explanation of the use of additional arguments.

FUNCTION DESCRIPTION:

The EBR command initiates execution of the command processor as the lead task in the batch task group previously created by the CB command. When the task group is dormant at the time the EBR command is issued, execution begins immediatly. When the group is not dormant, the request for execution is queued for execution when the task group becomes dormant (i.e., when the current batch request is terminated).

The command processor will first execute the EC file working-directory >START_UP.EC (if it exists). The working directory is the one specified in the optional -WD path argument. Whether or not these files exist, the command processor remains active expecting more commands.

Since the command processor obtains its commands from the file named in the in_path argument, that file must begin with a command, although it may contain other items (such as Editor directives) that the called command function may require for execution.

Example:

EBR CMND_IN -WD ^VOLA>JR

The batch task group is to be activated. It will receive its input from the file identified as CMND_IN; it will direct its output to the file CMND_IN.AO. The working directory will be ^VOLA>JR.

ENTER GROUP REQUEST

Command Name: EGR

Activate the lead task of an online task group previously created by a CREATE GROUP command.

FORMAT:

EGR id [in_path] [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of a task group previously created by a CG command specifying the same id.

[in_path]
The name of the file from which commands and user input are to be read by the task group during execution. This argument is set to null if not specified. It is required if the CG command specified the control argument -ECL.

[ctl_arg]
One or more control arguments chosen from the following list.

-OUT out_path

Defines the pathname of the file to receive user output (user_out file) and error output (error_out file) from the task group. If this argument is not specified, one of the followlng assumptions is made:

- If in_path specifies a disk file, out_path is in_path.AO
- If in_path specifies an interactive terminal, out_path is in_path
- If in_path is not specified, out_path is null
- If in_path specifies an input-only device, out_path is null

-WD path
Specifies that path is to be used as the working directory pathname. This argument is set to null if not specified.

-ARG arg arg . . . arg
Indicates that additional arguments required by the task group during execution follow. These additional arguments are passed to the lead task to be used as necessary, and are substituted for parameters in the command-in file. If used, the -ARG control argument must appear last. Refer to Appendix A for an explanation of the use of additional arguments.

FUNCTION DESCRIPTION:

The EGR command initiates execution of the lead task of a task group previously created by a CG command. If the task group is dormant when the EGR command is issued, task execution begins immediately. When the group is not dormant, the request for execution of the lead task is queued. Execution will take place when the task group becomes dormant. (This situation occurs when an earlier EGR command activates the task group and execution has not yet terminated.)

Execution of the lead task begins at the point implied by the -EFN argument.

When the command processor is the lead task, it first executes the EC file working-directory >START_UP.EC (if this file exists). The working directory used is that specified in the -WD path argument. Whether or not these files exist, the command processor remains active expecting more commands. After the START_UP.EC file is executed, execution begins

with the reading of the file named in the in_path argument. That file must begin with a command, although it may contain other items required for execution of the called command function.

Example:

```
EGR AX MPG_DATA -WD >UDD>SERVICES>SMITH -ARG '01/12/78 1100AM'
```

The task group identified as AX in a previous CG command is to be activated. The task group expects its input data to come from a file named MPG_DATA in the issuer's working directory; the group will write its output to a file named MPG_DATA.AO in some working directory. The working directory for group AX will be >UDD>SERVICES>SMITH. The lead task expects one argument, a date and time item. The item is enclosed in quotation marks because it contains an embedded space but is to be interpreted as a single argument.

ENTER TASK REQUEST

Command Name: ETR

Allocate and initialize a task request block and place it on the request queue of the indicated task.

FORMAT:

ETR lrn [ctl_arg]

ARGUMENT DESCRIPTION:

lrn
A logical resource number specified in a previous CREATE TASK command.

[ctl_arg]
One or more control arguments chosen from the following list.

-WAIT
Specifies that the command processor is to wait upon completion of the requested task before resuming execution.

-ARG arg arg . . . arg
Indicates that additional arguments required by the task during execution follow. These additional arguments are passed to the requested task to be used as necessary and are substituted for parameters in the command-in file. If used, the -ARG control argument must appear last. Refer to Appendix A for an explanation of the use of additional arguments.

FUNCTION DESCRIPTION:

The ENTER TASK REQUEST command is used to activate a task which was previously defined by a CREATE TASK command specifying the same logical resource number (LRN) as that named in this command.

The ETR command causes the construction of a standard task request block (TRB), which consists of the elements described in the *System Service Macro Calls* manual. Additional entries to accommodate task-specific arguments specified by the -ARG control argument are appended to the TRB as required.

Multiple tasks can be made to execute concurrently within a given task group by issuing multiple CT and ETR commands.

Tasks can also be made to execute serially; i.e., one task going to completion before a subsequent task begins execution. The -WAIT control argument is the mechanism that controls concurrency of execution. Judicious use of this argument can also result in a mixture of concurrent and serial execution (see example 3, below).

When all the created and requested tasks have terminated, the structures built by the CT and ETR commands can be removed by a BYE command. For a created group, a BYE command removes all structures other than those of the lead task (the command processor). After this action has occurred, the next group request, if any, can be honored. For a spawned group, the BYE command causes the group to be removed from the system.

In each of the following examples three tasks are assumed to have been previously created by the CT commands shown in the example in the description of the CT command, namely:

```
CT  10  2  -EFN   PROG10
CT  11  3  -EFN   PROG11
CT  12  2  -SHARE  10  ENTRY  2
```

Any other prerequisite commands (e.g., file creation, association of LFNs to pathnames) are also assumed to have been issued.

Example 1:

The three tasks are such that there are no dependencies among them, so they can be run concurrently. The following ETR commands are issued to activate them.

```
ETR  10
ETR  11
ETR  12
```

Example 2:

The three tasks are required to be executed in a particular sequence, determined by the order in which the ETR commands are issued. The following ETR commands are used to activate them.

```
ETR  10  -WAIT
ETR  12  -WAIT
ETR  11
```

In this case, execution of task 12 must await completion of task 10, and task 11 must likewise await completion of task 12. Since task 11 does not specify -WAIT, another (unrelated) activity can be initiated in parallel with the execution of task 11. Note, however, that if a BYE command follows the last ETR command, task 11 will probably not complete, since the BYE command takes effect even if a task within the task group is active.

Example 3:

Two of the tasks have a dependency between them and the third is independent of the other two. The following sequence of ETR commands can be used to activate them.

```
ETR  11
ETR  10  -WAIT
ETR  12
```

In this case, because task 11 does not specify -WAIT, both task 11 and task 10 are activated to run concurrently, but task 12 is dependent upon the completion of task 10. As in the previous example, another activity can be initiated concurrently with the execution of the third task.

EXECUTION COMMAMD

Command Name: EC

Invoke the command (EC) processor to read commands from a designated file.

FORMAT:

EC path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The name of a file, path.EC, that contains commands and EC directives.

[ctl_arg]
The list of additional character string arguments, arg arg . . . arg, that are to be substituted for substitutable parameters in the input lines of the command-in file. The pathname of the EC file is substituted for all occurrences of &0 in the command-in file, the first additional argument for all occurrences of &1, the second additional argument for all occurrences of &2, and so forth. Refer to Appendix A for a further explanation of the use of additional arguments.

FUNCTION DESCRIPTION:

The function of the command processor is to read from a previously created file a series of commands and EC directives. The command processor provides a mechanism whereby a sequence of routinely performed functions can be executed without the need for manually entering the commands.

The file whose name is path.EC is a sequentially processed file that contains the ASCII images of one or more commands and EC directives. These images are interpreted by the EC processor and acted upon as described in the following paragraphs.

When a command is encountered, it is passed to the command processor for interpretation and execution. The syntax of the command, as read from the file path.EC, must be identical to that which would have been entered from a terminal device, were the function to have been requested manually. All arguments and control arguments must be supplied as specified in the individual command descriptions.

When a command execution terminates, control is returned to the command processor, which then reads the next line from the file.

The EC file can also contain EC control directives that are not passed to the command processor, but are interpreted and acted upon by the EC directive routines. These directive lines are identified by a character string beginning with & and followed by a Δ (space or tab character). They provide control over certain operational aspects of the command processor and provide a degree of control over the logic of execution of the series of commands. Any ampersand directive other than those described below is treated as an &QΔ directive, except that a nonzero error status code is returned to the task that invoked the EC command (see the *System Messages* manual).

The EC control directives are described in detail in the following paragraphs.

&Δ
This directive signifies a comment line and is not further processed. The directive is visible to the user only if he obtains a listing of the EC file. The &Δ directive can be used (for example) to describe the function performed by the commands contained in the file.

&AΔ path
This directive causes the file specified by path to be attached as the user_in file. If path is omitted, the current command_in file is assumed to be the user_in file.

&DΔ

This directive restores the user-in file to that which existed when the EC file was invoked.

&FΔ

Command line printing is to be turned off; i.e., command lines are not to be written to the user-out file. This is the default; command lines are not normally written to the user-out file.

&NΔ

Command line printing is to be turned on. Each command line read from the EC file is written to the user-out file before being passed to the command processor. The & directives lines are not written.

&PΔ

The entire line, except for the &PΔ, is written to the user-out file. Printing of &PΔ lines occurs regardless of whether command line printing is on or off.

&G label1

This directive, in conjunction with the &L directive provides a "go to" capability, and in conjunction with the IF-THEN-ELSE directive provides a conditional execution of commands within the EC file. The next command to be processed is the command immediately after the first &L directive that defines the label.

&L label1

This directive defines a label that may be the object of a &G (or a conditional goto) statement. The label begins with the first non-blank (or tab) character after the &L and its length is restricted only by the inputn line length.

&IFΔ [EQUALΔ [RETCODE]Δhhhh] Δ&THEN [{ Δ&Q / Δ&G label1 }] [Δ&ELSE [{ Δ&Q / Δ&G label2 }]]

This directive causes the command processor to interrogate the status code (RETCODE) returned by the command executed immediately before the &IF directive. The subsequent processing depends on whether or not the status code entered by the user matches that returned by the previous command.

[EQUALΔ[RETCODE]Δhhhh]

Caution:
This argument including the double set of brackets must be entered when this directive is specified. This argument is an active function and the brackets are a part of it. Elsewhere, brackets denote optionality.

The hexadecimal number hhhh designates a status code. One to four digits may be entered. If less than four digits are entered, the field is left filled with zeros.

&THEN{ Δ&Q / Δ&G label1 }

If the status codes match, processing ceases or continues on the line following the label1 statement, depending on t the option selected. If the options are omitted, processing continues on the next line.

&ELSE{ Δ&Q / Δ&G label2 }

If the status codes do not match, processing ceases or continues on the line following the label2 statement, depending on the option selected. If the options are omitted, processing continues on the next line.

&QΔ

The execution of the current EC file is terminated, and control is returned to the invoking task. Implicit &QΔ directives may be executed, as described above, by invalid & directives

or because of error status codes returned by the interpretation or execution of a command line. To ensure proper termination of the EC command, every EC file should have the &QΔ directive as its last entry.

Example 1:

A user is developing a program named TEST. Several recursions of source unit correction, assembly, and link are required before the program is operational. The original source unit TEST.A has already been created using the Editor. An EC file PROG_DEV.EC has also been previously created, and contains the following commands and EC directives.

```
EC directives.
&ΔEDIT, ASSEMBLE, AND LINK PROGRAM 'TEST'
&PΔBEGIN EDITOR
&A
ED
(Editor directives) Δ
&IFΔ[EQUALSΔ&STATUSΔ0]Δ&THENΔ&ELSE&&GΔLABEL1
&PΔBEGIN ASSEMBLY
ASSEM TEST -COUT>SPD>LPT01
&IFΔ[EQUALSΔ&STATUSΔ0]Δ&THENΔ&ELSEΔ&&GΔLABEL1
&PΔBEGIN LINK
LINKER TEST -COUT>SPD>LPT01 -IN PATH1
&IFΔ[EQUALSΔ&STATUSΔ0]Δ&THENΔ&ELSEΔ&&GΔLABEL1
&PΔLINK COMPLETE
&GΔLabel2
&LΔLabel1
&PΔERROR ENCOUNTERED IN DEVELOPMENT SEQUENCE; EC TERMINATED.
&LΔLabel2
&QΔ
```

In order to execute the correction, assembly and link sequence the user has only to enter the command

```
EC  PROG_DEV
```

The command processor appends the .EC suffix to PROG_DEV and searches the current working directory for the resulting file name PROG_DEV.EC. Each command is preceded by an &PΔ directive which causes a typeout to the user-out file informing the user of his step-by-step progress through the sequence. The &IFΔ directives following each command line would cause an exit from the sequence if execution of the command resulted in an error reported by a non-zero error status code. The &P message after LABEL1 would be written to the user-out file, indicating to the user that some error condition has been detected.

Example 2:

Execution of the program 'TEST' created in example 1 above requires the creation of a work file for use by the program. This file is also to be deleted after the program is finished its execution. The following EC file, called EX_TEST.EC has been created to perform this sequence of functions:

```
&ΔEXECUTE PROGRAM 'TEST'
&PΔCREATE WORK FILE 'TEST01'
CF TEST01 -SEQ -SZ 100
&PΔEXECUTE 'TEST'
TEST arg1 . . . argn
FD TEST01
RL TEST01
```

```
&PΔEXECUTION OF 'TEST' COMPLETE
&QΔ
```

In this case, the user enters the command

```
EC  EX_TEST
```

which, in the same manner as in example 1, invokes the EC processor and turns control over to the sequence of commands and directives contained in the EC file. The work file TEST01 is created, the program 'TEST' is invoked, supplying any arguments (arg_1 . . . arg_n) which may be necessary to its execution, a dump of the work file is requested, and the file is then released (deleted).

EXPORT PAM FILE [MOD 400 ONLY]

EXPORT PAM FILE

Command Name: EX__PAM

Copy one or more sequential files to a BES1 and BES2 partitioned file.

FORMAT:

EX__PAM path pam [mem_1] ... [-R]

ARGUMENT DESCRIPTION:

path
The pathname of a directory containing one or more files to be exported.

pam
The name of the BES1 and BES2 partitioned file to which the sequential files are to be exported.

[mem_i]
The simple names of one or more files, immediately contained within path, to be copied to the partitioned file as members. If not specified, all files within path are copied.

[-R]
Indicates that if any member named by the mem_i argument already exists in the file named by pam, it is to be replaced.

FUNCTION DESCRIPTION:

The EX__PAM command permits the transfer of sequential files contained within the File System to BES1 and BES2 partitioned access method (PAM) files. Each sequential file, when copied to the PAM file, becomes a member in the PAM file, and has its name entered into the member index portion of the PAM file.

The path argument names the File System directory which immediately contains the files to be transferred; i.e., if the named directory contains subdirectories which themselves contain files, these latter files are not affected by the EX__PAM command. If no mem_i arguments are given, each file which is immediately subordinate to the directory named in the path argument is transferred. The pam argument names the BES1 and BES2 partitioned file which is to contain the transferred files. This file must have been previously created as a BES ½ offline activity. Three steps are involved in the creation of this file.

1. The volume which is to contain the PAM file can be initialized using the Initialize function of Utility Set 1 or can be created by GCOS 6. (If the volume has already been initialized and the file is to be added to the volume, this step is omitted.)
2. The file must be allocated using the Allocate function of Utility Set 1. Sufficient sectors must be allocated to contain the expected files and the accompanying member index.
3. The file allocated in step 2 must be initialized as a partitioned file using the Initialize function of Utility Set 1. The number of members specified must be large enough to accommodate the expected number of transferred files.

The user can refer to the *BES2 Utility Programs* manual (Order Number AU47) for full details regarding the allocation and initialization of partitioned files.

Because of the eight-character limit in the length of a BES1 and BES2 partitioned file member name, the first eight characters of each file to be transferred must be unique. File names longer than eight characters are truncated to eight characters.

Example:

```
EX__PAM  MYDIR  ^BESVOL>BESFIL  FILEA  FILEB  FILEC
```

Three files contained in the directory MYDIR are to be transferred to the partitioned file BESVOL>BESFIL. The files, FILEA, FILEB, and FILEC, are sequential files and are added as members to the previously created PAM file as new members.

FILE CHANGE

Command Name: FC

Change the contents of a disk sector or control interval.

FORMAT:

FC path

ARGUMENT DESCRIPTION:

path
Pathname of the file whose contents are to be changed. A peripheral device pathname indicates that sectors are to be changed. A file pathname indicates control intervals are to be changed.

FUNCTION DESCRIPTION:

The FC command is used to modify a file in accordance with directives submitted to the FC processor. A complete description of the FC directives and the file change function is presented in Appendix C.

FILE DUMP

Command Name: FD

Transfer the contents of the specified area of a disk or magnetic tape volume to the user-out file.

FORMAT:

FD path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the file or volume whose contents are to be dumped. The form of the pathname to dump a volume is

>SPD>dev_name[>vol_id]

If the vol_id is specified, the volume name is verified. To dump a file, a simple, relative, or absolute pathname may be used.

[ctl_arg]
One or more control arguments chosen from the following list:

{-FROM xx / -FM xx}
The dump starts with record xx which can be a decimal number or a hexadecimal number in the form X'hhhhh', where 'hhhhh' represents up to five hexadecimal digits.

{-LIMIT nn / -LI nn}
Dump the number of records specified by nn. If end of file is encountered before nn records are dumped, the dump terminates at end of file, nn can be a decimal number or a hexadecimal number in the form X'hhhhh', where 'hhhhh' represents up to five hexadecimal digits.

-CI
The dump is to be taken at the control interval level. If the path argument specifies a magnetic tape file or volume, the dump is taken at the physical block level. 7-track magnetic tape is only dumped in -CI mode.

-NWD
A magnetic tape volume is not to be rewound before opening it for the dump. This argument is valid only if -CI is also specified.

-BACK nn
A magnetic tape volume is to be backspaced nn blocks before dumping. This argument is valid only if -CI is also specified.

-HEX
Print only the hexadecimal representation of the file or volume content.

-ALPHA
Print only the ASCII representation of the file or volume content.

-OCTAL
Applicable only to 7-track magnetic tape. Print each frame from the tape as two octal characters.

-PACK
Applicable only to 7-track magnetic tape. File dump is to read the magnetic tape in 664 (packed) mode.

Note:
If no -HEX, -ALPHA, or -OCTAL argument is specified, hexadecimal and ASCII output will be printed.

FUNCTION DESCRIPTION:

The FILE DUMP command permits the user to obtain a printed listing of a disk or magnetic tape file or volume. The maximum number of characters of pathname that will appear on a heading depends on the number of options that appear on the header. Whether a file or a volume is to be dumped is determined by the form of the path argument. A disk file dump can be obtained by using any of the acceptable forms of a pathname. A disk volume dump or a dump of a magnetic tape file or volume is obtained by specifying the pathname in the form

>SPD>dev_name[>vol_id]

The dev_name portion of the path argument is the symbolic device name assigned at system building. The vol_id portion, if used, specifies that the volume name is to be verified before initiating the dump.

The output from the FD command is written to whatever file or device is currently assigned as the user-out file. For 7-track tape files, the entire contents of the tape are dumped (i.e., a physical dump occurs) subject to the -FROM and -LI argument values.

Example 1:

```
FD  MYFILE
```

A user file named MYFILE is to be dumped in its entirety. The output is written in both hexadecimal and ASCII representation.

Example 2:

```
FD  >SPD>MTU01  -NWD  -BACK  5  -LIMIT  10  -CI
```

A portion of a magnetic tape volume is to be dumped. The volume is that defined at system building as MTU01. Ten physical blocks are to be dumped after backspacing the tape five blocks. The -NWD argument is used to prevent rewinding of the tape, and, in conjunction with the -BACK and -LIMIT arguments, effectively causes a dump of five blocks on either side of the tape's current position.

FILE OUT

Command Name: FO

Change the destination to which user output is sent.

FORMAT:

FO [path]

ARGUMENT DESCRIPTION:

[path]
The name of the new user-out file. If this argument is omitted, the user-out file reverts to that established at task group initiation.

FUNCTION DESCRIPTION:

The FILE OUT command defines a new device or file to which user output generated by a task is written. The file or device is reserved with exclusive concurrency except that the operator terminal is reserved with shared read/write access. When a task group is initiated, the file which is to receive this output is established by the -OUT control argument of the ENTER GROUP REQUEST, ENTER BATCH REQUEST, or SPAWN GROUP command. Error output is also written to the same file. The FO command makes it possible for a series of group requests to write their output information to separate files or devices. It does not affect the destination of error output; this is always written to the originally defined file. The user of the FO command with no argument resets the destination of user output to that of error output as defined in the EGR, SG, or EBR command.

Example:

FO REPORT_OUT

The output generated by the issuing task is to be redirected to a file named REPORT_OUT in the working directory.

FORTRAN

Command Name: FORTRAN

Compile the FORTRAN source program unit represented by the indicated file name, applying the specified compilation options.

FORMAT:

FORTRAN path [ctl_arg]

ARGUMENT DESCRIPTION:

path

Pathname of the source unit file to be compiled. Omit the suffix (.F).

[ctl_arg]

None or any number of the following control arguments can be entered, in any order.

-AS

Output is assembly language text contained in the file path.A. This file can be used with the -SAF option as input to the Assembler. Modifications may be required to assemble the file with the -LAF option (see the *Assembly Language* manual).

-COUT out_path

Listing will be written to the file out_path; a suffix is *not* appended to the file name. If this argument is omitted, the listing will be written to the file path.L in the working directory.

-FS

The compiler will not define the size of the work area when compiling a subroutine, nor will it cause the implicit initialization of the work area when the subroutine is executed. See note 3, below.

-HS

The source unit comprises Hollerith code or the source unit was created using a Series 200/2000 or Model 716 Central Processor.

{-LIST_ERRS
-LE }

Specifies that only those source lines containing compilation errors, together with their error codes, are to be listed.

Default: If omitted, and -NL is not specified, the complete source program is listed, including error codes, if any.

{-LIST_OBJ
-LO }

List object output. Object text listings in assembly language format will be interspersed with source text listings.

Default: Object text is not listed.

Note:

This argument is not meaningful when used with the -AS argument.

{-NO_LIST
-NL }

Suppress all listings.

Default: If omitted and -LE (or -LIST_ERRS) is not specified, the complete source program is listed, including error codes (if any).

{-NO_OBJ
-NO }

Suppress generation of the object text unit. (This option should not be used with the -AS argument.)

Default: If omitted and -AS is not specified, an object text unit is produced as file path.O.

-SI
One word is allocated for each integer and logical variable (short integer and logical variables).

Default: Two words are allocated for each integer and logical variable.

Note:
This argument affects space allocation only. The range of integer and logical variables is the same regardless of whether the argument is specified.

{-SIZE nn
-SZ nn }
nn designates the maximum number of 1024-word blocks of memory that the compiler can use for tables. If the requested amount of memory is not available, the compiler will use the available amount of memory.

Default: One block (1024 words).

-UC
Suppress generation of embedded links to any subroutines referenced by a CALL statement, and functions other than intrinsic functions.

-UZ
Suppress generation of embedded links to system subroutines (i.e., all subroutines beginning with the letters ZF).

-WRK n
Establishes the size in words of the object me workspace for FORTRAN programs; n specifies the number of words and must be a 1- to 4-digit decimal number from 1 through 9999. See Note 3 below.

Default: 356 words.

{-LAF
-LA }
Specifies that long address form (LAF) object text is to be generated.

Default: Short address form (SAF) object text is generated.

Note:
This argument is not meaningful when used with the -AS argument.

FUNCTION DESCRIPTION:

The FORTRAN command is used to invoke the FORTRAN Compiler component.

The path argument can assume any of the acceptable forms of a pathname, although normally it would be a simple name, indicating that a source program unit which resides in the working directory is to be compiled. Wherever it exists, it must be suffixed with a .F suffix, indicating that it is a FORTRAN language source unit. The path argument must be given *without* the .F suffix; the compiler appends the suffix prior to searching the directory for the source unit.

If the -COUT control argument is not specified, the source and object listings (if specified) are written to a file created by the compiler in the working directory, having a file name of the form path.L, the path portion being the last (or only) element specified in the path argument. This file can be subsequently listed on a line printer by using the PRINT utility command. If a different file is specified by using the -COUT argument, the listings are written to the file whose pathname is out_path. The compiler does *not* append an .L suffix.

The object text unit generated by the compiler is written to a file whose name is of the form path.O, and is contained in the working directory.

If files of the form path.L and path.O already exist in the working directory, they are overlaid by the output generated by the current compilation.

Notes:

1. Either LO or NL may be specified, but not both. If neither is specified, the compiler produces a listing of the source text and diagnostics.
2. The FORTRAN Compiler always issues a typeout, of the number of errors found, to the error-out file.
3. Most FORTRAN programs call input/output routines and intrinsic functions, the majority of which utilize a workspace. Prior to the invocation of any one of these modules (routines or functions) the workspace must be initialized. The FORTRAN compiler automatically generates a workspace declaration and the prologue code for initialization of the workspace in each main program and each subprogram for which the -FS argument is not specified (the -FS argument is ignored if specified for a main program). Use of the -FS argument implies that the subprogram either does not need the workspace or depends upon another program to declare and initialize the workspace.

   To avoid execution time errors involving use of the workspace, the declarations of modules linked together in a bound unit must be identical. Therefore, if you want to create general purpose subroutines to be used in applications which require workspace areas of various sizes, you should compile the subroutines with the -FS argument.

   In some applications, variations in the workspace size may be necessary to increase or decrease the default imput/output buffer space of 128 words. The -WRK argment is used to make this modification. For details, refer to the *FORTRAN Reference* manual.
4. The compiler is designed for batch compilations. That is, many source modules can be passed to the compiler under one file name and each source module will be compiled separately. The compiler expects an END statement in each source module, followed be either an end-file or a new source module. In addition, the SI, WRK, FS and HS arguments can be passed to the compiler as part of the source module, rather than as arguments of the FORTRAN command. Arguments specified in the FORTRAN command apply to all source modules in the batch. Including these arguments in the source module permits them to be varied on a module-by-module basis. To include these arguments in a source module, enter each as a special comment immediately following the PROGRAM, SUBROUTINE or FUNCTION statement for the module. The general form is:

$$C^*OPT = \begin{cases} SI \\ WRK = n \\ FS \\ HS \end{cases}$$

   For example,

   C*OPT = WRK = 400

   Note that the work area size argument requires an equals(=) sign when it is specified as part of a source module. More than one argument can be specified, but each requires a separate comment line.

Example:

```
FORTRAN FTPROG -LIST_OBJ -SI -COUT >SPD>LPT01
```

A FORTRAN source program, FTPROG.F, residing in the working directory, is to be compiled. The source and error listings are to be written to the printer LPT01, and the object text unit is to be written to the file FTPROG.O in the working directory. Short-form integer and logical variables are to be generated.

GET FILE

Command Name: GET

Reserve a file (i.e., a tape or disk file, a disk directory, a card reader, a printer, or a terminal device). Establish an association between the reserved file and a logical file number (LFN) if such an association has not already been established.

FORMAT:

Format 1 (Disk Files):

GET [path] [lfn] [disk file ctl_arg]

Format 2 (Labeled Tapes, Card Readers, Printers, and Terminal Devices):

GET [path] [lfn]

Format 3 (Unlabeled and Output Tape Files):

GET [path] [lfn] [unlabeled/output tape file ctl_arg]

Format 4 (Disk Directories):

GET [path] [lfn] [disk directory ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the file being reserved; can be any valid file- or device-level access pathname. This argument is not required if the lfn argument is supplied and a relationship between path and lfn has been established previously.

lfn
Logical file number (LFN) by which the file is to be referenced during access. When supplied, the lfn argument must be a decimal value in the range 0 through 255. Within a task group, an LFN can be used to access only one pathname at a time.

disk file ctl_arg
Any, all, or none of the following may be specified in any order:

-MOUNT
Specifies that a mount request should be displayed on the operator terminal if the required volume is not mounted. If this argument is not specified, and the file or volume is not mounted, a 020C (volume not found) message will be displayed.

-NBF n
Specifies the number of buffers to be allocated for I/O when the file is opened for access. This argument applies only to disk files and volumes that are accessed at the record level. The default is one buffer for files whose organization is other than indexed sequential and two buffers for indexed sequential files.

-LOCK
Indicates that record locking is to be performed when this file is accessed. A 022E (record lock concurrency conflict) message is issued if the file is already in use without locking, and reservation is denied. Once a file is reserved with locking, it cannot be reserved by another user unless this argument is specified.

-LRSZ n
Specifies the logical record size (in bytes) for fixed-relative files without deletable records; n is a decimal number from 0 through 65535. The default is the value specified when the file was created.

-ACCESS {R | W} -SHARE {R | W | N}
Specifies the desired file access concurrency (i.e., how the task group intends to access the

file and in what way it is willing to share access to the file with other task groups). None, one, or both of the following keywords can be specified.

-ACCESS { R / W }

Indicates how the task group intends to access the file. R mans read access; W means both read and write access. If the keyword is omitted, or if only the keyword is specified (R or W omitted), the default is W.

-SHARE { R / W / N }

Indicates how the task group is willing to share the file (i.e., what it will allow other users in other task groups to do concurrently). R means read access only; W means both read and write access; N means neither read nor write access. If the keyword is omitted, or if only the keyword is specified (R, W, or N omitted), the default value is R.

unlabeled/output tape file ctl_arg

Any, all, or none of the following may be specified, in any order:

-FSN { * / n }

Provides the tape file sequence number that indicates the relative position of an existing file on an ANSI tape volume. The value of n can be any decimal number from 1 through 254, specifying the relative position of the file on the volume. If -FSN * is specified, the entire volume is searched for the file. The default condition is that the desired file is next on the volume, relative to the current position of the volume.

-BKSZ n

Specifies the block size (in bytes). For files with fixed length records, n is a multiple of the record size plus the 6-character binary block sequence number (if specified). For files with variable length records, the block size can be any value, but should be at least as large as the maximum record size plus the 4-character logical record header and the 6-character binary sequence number (if specified). n is a decimal number in the range 0 through 65535. Computation of the default value depends on the -LRSZ, -TDF, and -BSN argument values. See Figure 2-2.

-LRSZ n

Specifies the logical record size (in bytes). For files with variable length records, this is the maximum record size. This value does not include the 4-byte logical record header; it includes only the data portion of the record. n is a decimal number in the range 0 through 65535. Computation of the default value depends on the -BKSZ, -TDF, and -BSN argument values. See Figure 2-2.

-TLF fmt

Indicates the tape label format. The value for fmt must be one of the following:

STD — Standard labels

NONE — Unlabeled

The default value is STD.

-TDT type

Indicates the tape data type. The value for type must be one of the following:

ANSI — ANSI level 3

H — Honeywell

The default value is ANSI.

-TDF fmt

Indicates the tape data format. The value for fmt must be one of the following:

F — Fixed length records
D — Variable length records (decimal size count)
U — Undefined records

The default value is D.

-BSN
Indicates that each block on the tape has a 6-character block sequence number in the first six characters of the block. For input, a block sequence number is assumed to be present. For output, a block sequence number will be inserted. If this argument is omitted, block sequence numbers are neither expected when reading nor inserted when writing.

-TRP n
Indicates the tape retention period in days. n is a decimal value from 0 through 65535.

The default value is 0, meaning no retention period.

disk directory ctl_arg
Both of the following may be specified; -SHARE N must be specified.

-MOUNT
Specifies that a mount request should be displayed on the operator terminal if the required volume is not mounted. If this argument is not specified, and the file or volume is not mounted, a 020C (volume not found) message will be displayed.

-SHARE N
Users in other task groups have neither read nor write access to this directory.

FUNCTION DESCRIPTION:

The GET command is used to reserve a tape file, disk file, disk directory, card reader, printer, or terminal device. The reservation is made at the task group level. The GET command stays in force until negated by a REMOVE command. Get File ($GTFIL) and Remove File ($RMFIL) macro calls that are executed by programs run during the period in which the GET command is in force do not affect the conditions established by the GET command.

If a file is referred to by more than one LFN, a GET command is required for each LFN. A GET command can be used to change attributes established by a previous GET command for the same LFN, provided the file is not currently opened in the same task group from which the GET is issued. Establishment of all reservation attributes must precede the opening of the file.

If an LFN is not supplied, the pathname is reserved for the task group. The LFN must be supplied by a Get File ($GTFIL) macro call issued by the task that will access the file.

If the volume id portion (root) of the pathname identifies a volume that is mounted, but a subordinate directory (or the file name itself) is not found on the volume, a 0209 (named file or directory not found) message is displayed, regardless of the specification of the -MOUNT argument.

If an operator terminal is not included in the system, or if messages to the operator terminal have been suppressed (through a $CMSUP macro call), a GET command issued to reserve a volume that is not mounted results in the display of a 020C (volume not mounted) message on the user terminal.

By increasing the number of buffers, I/O time can be significantly reduced for a file being accessed randomly. When accessing a record of a file, the File System first checks all buffers allocated for the file to determine whether the desired record is already in memory.

The record lock facility requires added effort on the part of the user when programs that share files are constructed. The -LOCK argument alone is not sufficient to guarantee interference protection. For a more detailed discussion of record locking, refer to the $GTFIL and $ULREC macro calls in the *System Service Macro Calls* manual.

The -LOCK argument allows the user to initiate multiuser interference protection mechanisms for shared file access. When more than one user (task group) shares access to a file, or

when cooperating tasks within a task group share access to a file, the task groups and tasks can be protected from interferring with one another when they attempt to access the same area of the file.

Lock requests are valid only for disk resident files. Directories and device files cannot be reserved with lock. Files reserved with lock can only be opened for access at the record level.

If the user who first reserves the file requests locking, all subsequent reservations of the file must also request locking. If the user who first reserves the file does not request locking, no subsequent reservation of the file can request locking unless the concurrent access option specifies read access only, with read/write sharing (-ACCESS R -SHARE W). The user who makes this specification can read the file but cannot write into it. Further, no data integrity is guaranteed for his read operations.

The -ACCESS and -SHARE arguments are indicators of how the user will access the file and to what degree he will allow others to share the file concurrently. The concurrency arguments are more explicitly described below.

Argument	**Meaning**
-ACCESS R	I will read only
-ACCESS W	I will read and write
-SHARE R	Others can read only
-SHARE W	Others can read and write
-SHARE N	Others can neither read nor write
Omitted	If the file is already reserved, the last concurrency specified is used. If the file is not already reserved, -ACCESS W -SHARE N is used.

Although there are a number of combinations of the concurrency arguments, some of which produce identical results, the forms shown below describe all concurrent access possibilities.

Argument(s)	**Meaning**
-ACCESS R	I will read; others can read; no one can write
-ACCESS R -SHARE W	I will read; others can read and write
-SHARE N	I will read and write; others can do nothing
-ACCESS	I will read and write; others can only read
-SHARE	I will read and write; others can read and write
Omitted	If the file is already reserved, the last concurrency specified is used. If the file is not already reserved, -ACCESS W -SHARE N is used.

If a directory is reserved exclusively (i.e., with the -SHARE N argument), all subdirectories and files inferior to the directory are held exclusively. For example, GET ^volid -SHARE N reserves the entire volume for exclusive use by the requesting task group.

The concurrency arguments do not apply to disk volume (device-level) reservations. If the pathname is of of the form >SPD>dev_name>volid, the reservation is performed as though -ACCESS R -SHARE W had been specified. If the pathname is of the form >SPD>dev_name, the reservation is performed as though -SHARE N had been specified.

The concurrency arguments do not apply to tape file or volume reservation. Regardless of the pathname form used, the reservation of tapes is always performed as though -SHARE N had been specified.

With the exception of -FSN, tape-specific arguments apply only if the tape is being opened (see the $OPFIL macro call in the *System Service Macro Calls* manual) in RENEW mode or the file is unlabeled. (In these cases, there are no labels.) If labels are present and the file is opened in PRESERVE mode, the label contents override the arguments supplied in the GET command.

When a tape file name is specified, an existing file is to be opened; the file name is checked in the following manner:

- If the -FSN argument is not specified, the File System checks the next file on the volume for a matching file name.
- If the -FSN argument is specified in the form -FSN n (where n is a decimal number from 1 through 254), the File System checks the nth file on the volume for a matching file name.

  Note:

  When the -FSN n argument is specified, the tape volume is positioned to the nth file before the file name is checked.
- If the -FSN argument is specified in the form -FSN *, the File System checks all files on the tape for a matching file name, beginning at the current position of the volume.

The maximum file name length is 12 characters.

For tape files, the default block size (-BKSZ) and logical record size (-LRSZ) are computed as shown in Figure 2-2. When the block and record sizes are supplied, they are checked for validity as shown in Figure 2-3.

Example 1:

```
GET ^BOOKS>FILEB  22  -MOUNT  -NBF  2  -ACCESS  W  -SHARE  N
```

The file whose pathname is ^BOOKS>FILEB is reserved and associated to logical file number 22. If the volume is not mounted, a mount message is issued. Two buffers are allocated. All tasks in this task group have read and write access; tasks in other task groups cannot share the file. Therefore, if the file is already reserved for another task group, reservation will be denied. Otherwise, the file is reserved and reservation requests by any other task group will be denied.

Example 2:

```
GET  ^BOOKS>FILEB  -SHARE  W  -LOCK
```

The file whose pathname is ^BOOKS>FILEB is reserved. The pathname and an LFN have previously been associated. Tasks in this task group have the default read/write access; tasks in other task groups may also be granted read or write access. Record locking is in effect for access to this file. If this is the first request for the file, no other task group or task in this group can reserve the file unless it specifies -LOCK. If the file is already reserved without -LOCK having been specified, this request will be denied.

Example 3:

```
GET  ^BOOKS>FILEB  30
```

The file whose pathname is ^BOOKS>FILEB and logical file number 30 are associated. If this file reservation follows a previous reservation, with the same pathname, the reservation options do not change. If this file is being reserved for the first time, it can be read from or written to in this task group, but it is not shared by other task groups.

Example 4:

```
GET  >SPD>DSK03>VOL001  12
```

A disk volume whose pathname is >SPD>DSK03>VOL001 is reserved and associated with logical file number 12. The disk volume can be read by this task group and can be read from or written to by other task groups.

Example 5:

```
GET >SPD>MT902>VOL3>FILE_2 -FSN * -BKSZ 326 -LRSZ 80 -TDF F -BSN
```

The magnetic tape file whose pathname is >SPD>MT902>VOL3>FILE_2 is reserved. The entire tape volume is to be searched for FILE_2. FILE_Z is a fixed length sequential file whose block size is 320; the value 326 is specified for -BKSZ since the tape has a block sequence number (-BSN). The logical record size is 80; there are 4 records per block. The tape has standard labels; its data type is ANSI; and its data format is fixed length records.

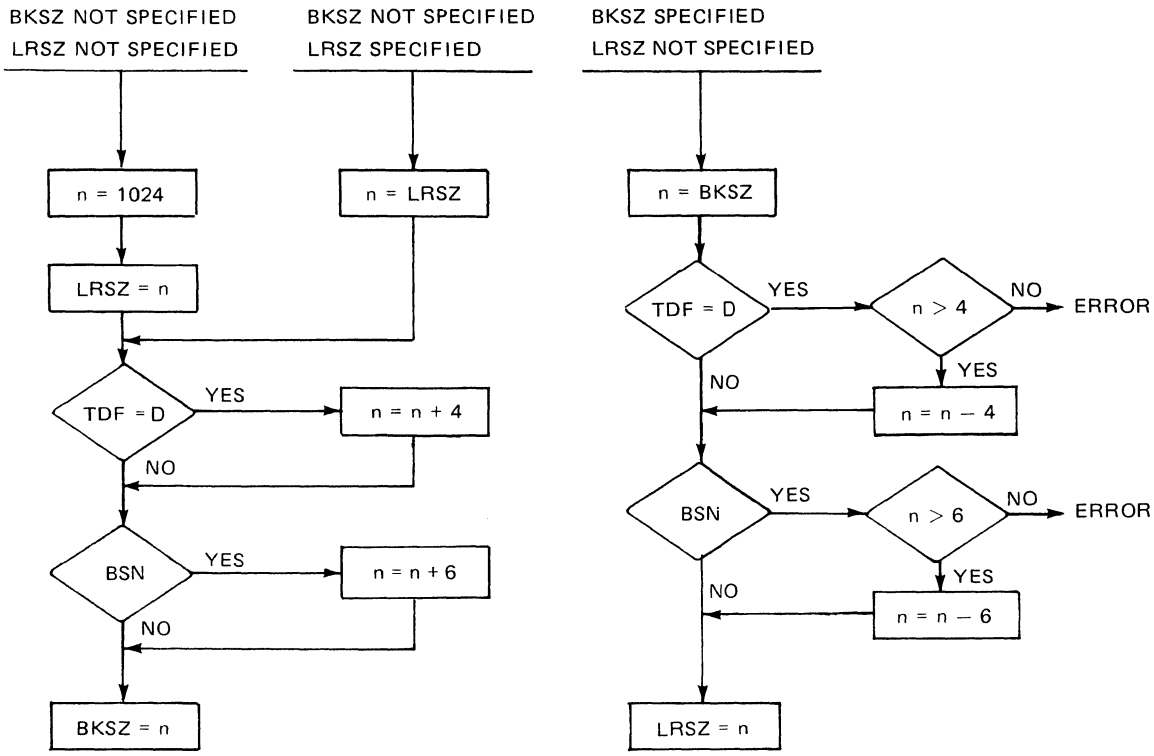


Figure 2-2. Default Block and Logical Record Size Calculation

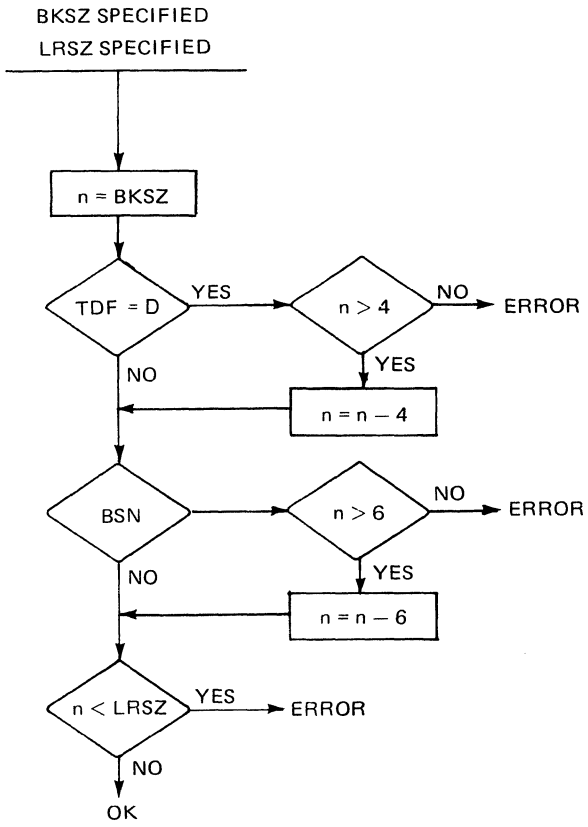


Figure 2-3. Block and Logical Record Size Validity Checking

IMPORT PAM FILE
[MOD 400 ONLY]

IMPORT PAM FILE

Command Name: IM_PAM

Transfer one or more BES1 and BES2 partitioned file members to the File System.

FORMAT:

IM_PAM pam path [mem$_i$] . . . [-R]

ARGUMENT DESCRIPTION:

pam
The name of the BES1 and BES2 partitioned file from which members are to be transferred.

path
The name of a directory into which the BES1 and BES2 file members are to be transferred.

[mem$_i$]
The names of one or more partitioned file members which are to be transferred to the File System.

[-R]
Indicates that if a file named by a mem$_i$ argument already exists in the directory named by path, it is to be replaced.

FUNCTION DESCRIPTION:

The IM_PAM command permits the transfer of one or more BES1 and BES2 partitioned access (PAM) file members into the File System. A PAM file member, when transferred, becomes a variable sequential file contained within the directory specified by the path argument.

The pam argument names the PAM file which contains the members to be transferred and has the format ^volid>partitioned_file_name. The file has been previously created using BES1 and BES2 offline procedures.

The path argument names the File System directory which is to contain the files transferred to it. It must have been previously created through the use of the CREATE DIRECTORY command.

Each of the mem$_i$ arguments, if any are specified, names a member of the PAM file, specified by the pam argument, which is to be transferred to the File System. If no mem$_i$ arguments are specified, every member contained in the file is transferred and becomes a sequential file.

If the -R control argument is not specified, and if a GCOS 6 file whose name is specified by a mem$_i$ argument already exists in the file system, an error message is issued.

Example:

IM_PAM ^BESVOL>BESFIL MYDIR MEM01 MEM02 MEM03 -R

Three PAM file members, MEM01, MEM02, and MEM03, contained in the partitioned file ^BESVOL>BESFIL, are to be transferred into the File System directory MYDIR. If the file system already contains any files whose names are the same as those of the specified members, they are to be replaced.

INVOKE RBT TASK GROUP

Command Name: RBT

Invoke a remote batch terminal (RBT) task group and associate it with a logical stream.

FORMAT:

RBT lrn

ARGUMENT DESCRIPTION:

lrn
The logical resource number (defined at system building) that specifies the stream to be used for remote batch operations.

FUNCTION DESCRIPTION:

See the *System Building* manual for further details on the use of this command. A complete description of remote batch operations is given in the *Remote Batch Facility User's Guide.*

ISL CONFIGURATOR [MOD 400 ONLY]

Command Name: ISLCON

Generate an ISL loader to load Intersystem Link (ISL) address maps and masks.

FORMAT:

ISLCON path [{-SAF / -LAF}]

ARGUMENT DESCRIPTION:

path
The pathname of the volume for which the loader file is to be written.
For SAF mode, pathname is ^volume_name>ISL_ROUTINES.
For LAF mode, pathname is ^volume_name>ISL_ROUTINEL.

FUNCTION DESCRIPTION:

The ISLCON command is used to invoke the ISL Configurator which obtains ISL configuration directives from the user-in file. It generates an executable loader program that will load ISL address maps and masks during ISL configuration. The ISL loader is written to the file specified in the "path" argument. A description of the ISL directives is contained in Appendix B of this Manual.

Example:

ISLCON ^V20022 -SAF

The ISL loader is to be built. ISL configuration directives are obtained from the user input file and written to the file ^V20022>ISL_ROUTINES.

LINKER

Command Name: LINKER

Create a bound unit from one or more object text units, applying the specified options.

FORMAT:

LINKER name [ctl_arg]

ARGUMENT DESCRIPTION:

name

Pathname of the created bound unit file. The pathname can be a simple, relative or absolute name. If the specified file already exists, it is overlaid with the new bound unit.

[ctl_arg]

One or more control arguments chosen from the following list:

-IN path

Specifies the pathname of the file containing the linker directives. The file can be read from a disk device, a card reader, or a terminal device. If not specified, the file named in the in_path argument of this task group's EGR, SG, or EBR command is used.

-COUT out_path

This argument is available for use only when processing under Mod 400 operating system software.

Specifies the name of the file to which the map listing will be written. The map listing can be written to a disk device, a printer, or a terminal device. If this argument is not specified, the listing is written to the file name.M in the same directory as the output file, overlaying any existing file by that name.

{-SAF / -LAF / -SLIC}

When processing under Mod 600 operating system software, the bound unit always executes in LAF mode.

Indicates the addressing mode in which the bound unit is to execute. -SAF indicates short (1-word address form); -LAF indicates long (2-word address form); -SLIC indicates the bound unit is to be capable of executing in both SAF and LAF mode.

{-SIZE nn / -SZ nn}

This argument is available for use only when processing under Mod 400 operating system software.

Designates the maximum number of 1024-word memory modules available for the Linker symbol table; nn must be from 01 through 32. If this argument is not specified,

-VERBOSE

All directives are output to the list file.

-NOMAP

Suppresses the generation of the list file.

-PT

Prompt character requested. If this argument is omitted, no prompt character is given.

-W

Work files are not to be deleted. If this argument is omitted, work files are deleted when no longer required.

-R

Code and data are to be segmented. If this argument is omitted, no segmenting of code and data is performed.

FUNCTION DESCRIPTION:

A complete description of the Linker is provided in the *Program Execution and Checkout* manual.

LIST ACCESS CONTROL LIST

List entries on access control list of a file or directory.

Command Name: {LA | LIST_ACL}

FORMAT:

{LA | LIST_ACL} [path][user_id] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the pathname of a file or directory. If this argument is omitted or if -WD is entered, the working directory is specified. If it is omitted, user_id cannot be specified.

user_id
Specifies an access control name that must be of the form person.account.mode. All ACL entries with matching names are listed. (For a description of the matching strategy, refer to the SET_ACL command.) If user_id is omitted, the user's person.account.* is used.

ctl_arg
One or more control arguments from the following list.

-A
-ALL
Causes the entire ACL to be listed. If user_id is omitted, -A or -ALL need not be specified. If user_id is specified and -A or -ALL is also specified, the entire ACL is listed.

-BF
-BRIEF
Suppresses the message Δ"USER NAME NOT ON ACL"

FUNCTION DESCRIPTION:

The LIST_ACL command causes the entries on the access control list (ACL) of a file or directory to be listed. To use the command, the user must have list access to the directory containing the ACL.

If the command is invoked with no arguments, the entire ACL on the working directory is listed.

If user_id is specified (and -A or -ALL is not), the ACL entries that match the access control name are listed.

Example:

```
LIST_ACL  ^SYS01>UDD>PROJ1>FILEAA
```

The entire ACL contained on FILEAA is listed.

LIST COMMON ACCESS CONTROL LIST

Command Name {LCA | LIST_CACL}

List the entries on the common access control list (CACL) in the specified directory.

FORMAT:

{LCA | LIST_CACL} [path] [user_id] [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the directory containing the CACL entries. If this argument is -WD, or if this argument is omitted, the working directory is used. If the argument is omitted, user_id cannot be specified.

user_id
An access control name having the following format:

person.account.mode

All CACL entries with matching names are listed. If user_id is omitted, all CACL entries are listed.

[ctl_arg]
The following optional control arguments can be chosen:

{-A | -ALL}

Causes all entries on the requested CACL to be listed. If user_id is omitted, this argument is redundant; if both are specified, this argument overrides user_id.

{-DIR | -DIRECTORY}

Specifies that directory CACL entries are to be listed. If -FILE (see below) is also specified, both file and directory CACL entries are listed. If neither -DIR nor -FILE is specified, all CACL entries are listed.

-FILE
Specifies that file CACL entries are to be listed. If -DIR is also specified, both file and directory CACL entries are listed. If neither -DIR nor -FILE is specified, all CACL entries are listed.

{-BF | -BRIEF}

Suppresses the message "USER NAME NOT ON CACL"

FUNCTION DESCRIPTION:

The LIST COMMON ACCESS CONTROL LIST command causes the entries on the common access control list (CACL) in the specified directory to be listed. Directory CACL entries, file CACL entries, or both, can be listed. The user must have list access to the directory referred to by path.

If this command is invoked with no arguments, all CACL entries on the working directory are listed.

If user_id is specified (and -A or -ALL is not), the CACL entries that match the access control name are listed. An explanation of the way in which names are matched is given in the description of the SET_ACL command.

Example:

```
LIST_CACL  -WD JONES.INTFIN.*  -DIR
```

The directory CACL entries in the working directory that have JONES as the person, INTFIN as the account and any value as the mode are listed.

LIST CREATION DATE

Command Name: LCD

List the creation date of a named file or of selected files in a named directory.

FORMAT:

LCD path

ARGUMENT DESCRIPTION:

path
Pathname of file or directory. The pathname can be absolute or can be relative to the working directory. This argument can also be a star name designating a group of files (see Section 1).

FUNCTION DESCRIPTION:

The LCD command is used to list the creation dates of source files, object files, list files, and bound units. The command is applicable only to files with creation dates. If the path argument refers to a directory, only those files with creation dates will be listed.

The information provided by the LCD command depends on the type of file, as follows:

Type of File	Information Listed
Source	Revision operand and comment line from TITLE statement
Object	Address mode, revision operand, assembly date/time, and Assembler identification
List	Address mode, revision operand, assembly date/time, and Assembler identification
Bound unit	Creation date/time and address mode

Examples:

```
LCD  ALPHA.A
```

List the creation date of the assembly language source file ALPHA.

```
LCD  >UDD>JOHNX1>PROG1
```

List the creation date of the bound unit PROG1.

```
LCD  ^VOL01>UDD>CALDERA
```

List the creation dates of all files contained in directory CALDERA that have creation dates

```
LCD  **
```

List the creation dates of all files contained in the working directory that have creation dates.

```
LCD  *.O
```

List the creation dates of all object files in the working directory.

LIST DATA EXCHANGE (IBM)

Command Name: LSDE

List by file name the contents of an IBM diskette.

FORMAT:

LSDE [ctl_arg]

ARGUMENT DESCRIPTION:

[ctl_arg]
Arguments accompanying the LSDE command are listed below.

-PN
Define the directory from which the entries are to be listed. Only one path name may/need be specified since an IBM diskette has only one directory. -PN is followed by the pathname.

Note:

-PN must be present when a path name is included and must be followed by that pathname. In the case of LSDE the path *cannot* be defaulted, and it must be of the form:

>SPD>DKSxx>volid>data_set_name

-BF
Print only a brief list of the contents of the specified diskette.

-DTL
Print a detailed list of contents of the specified diskette.

Note:

If neither the -BF nor the -DTL arguments are specified, a normal listing is printed (see examples).

FUNCTION DESCRIPTION:

The purpose of the LSDE command is to make it possible to list, by file name, the files contained on a 3740-like diskette. Having only a single directory, the entire contents of the volume can be listed with the path name specified only once.

EXAMPLES:[1]

(Normal List)
VOLUME: xxxxxx. . .(path of DE volume)
data set name: xxxxxx

boe	eoe	eod
xxxxxx	xxxxxx	xxxxxx

[1] where:
x = a decimal integer
boe = beginning of extent
eoe = end of extent
eod = end of data
type = file type (s= sequential)

Multi-Vol stipulates whether the file is contained on a single volume or more than one. A value of "s" indicates that the file is on a single volume. A value of "c" indicates that the file is continued on another volume. A value of "l" indicates that this is the last volume of a multiple volume file.

"Vol seq" indicates the volume sequence number:

01= first volume
02= second volume

data set name: xxxxxx

boe	eoe	eod
xxxxxx	xxxxxx	xxxxxx
		.
		.
		.
		.

data set name: xxxxxx

boe	eoe	eod
xxxxxx	xxxxxx	xxxxxx

(Brief List)
VOLUME: xxxxxx. . .(path of DE volume)
data set name: xxxxxx
xxxxxx
xxxxxx
.
.
.
.
xxxxxx

(Detailed List)
VOLUME: xxxxxx...(path of DE volume)
data set name: xxxxxx

boe	eoe	eod
xxxxxx	xxxxxx	xxxxxx

type	sector size	rec size
x	xxxx	xxx

multi vol		vol seq
xx		xx

create date		exp date
xxxxxx		xxxxxx

LIST NAMES

Command Name: LS

Display the names of one or more elements contained in the specified directory, along with their types, attributes, and sizes.

FORMAT:

LS [ctl_arg] [entry_name]

ARGUMENT DESCRIPTION:

[ctl_arg]
One or more control arguments chosen from the following list:

-PN path
Specifies the directory from which entries are to be listed. If this argument is omitted, entries contained in the working directory are listed. Can be any valid form of pathname; can use the star name convention (see Section 1).

-FILE
Indicates that only file entries are to be displayed. If none of the control arguments -FILE, -DIR, or -ALL are specified, -ALL is the default.

-DIR
Indicates that only directory entries (i.e., directories subordinate to the specified directory path) are to be displayed.

-ALL
Indicates that both file and directory entries subordinate to the specified directory are to be displayed.

-BRIEF
-BF
List only the name, type and total number of sectors of each entry are to be displayed.

{-DETAIL
-DTL }
Specifies that the file type and attributes of each entry are to be displayed.

[entry_name]
Specifies the name of the entry to be displayed. If this argument is omitted, all entries of the type(s) specified by control arguments are displayed.

FUNCTION DESCRIPTION:
The LIST NAMES command permits the user to obtain a listing of the file and/or directory entries contained within a given directory. It also displays various attributes of file entries: file type, starting sector, number of sectors, and record length.

The following list gives the possible file type designators and their meanings.

Type	Meaning
R1	BES fixed relative, static allocation, no deletable records.
R2	BES fixed relative, dynamic allocation, no deletable records.
R4	BES fixed relative, static allocation, deletable records.
R5	BES fixed relative, dynamic allocation, deletable records.
D	Directory.
S	Variable sequential.
R	Relative.
ID	Indexed (data area).
I	Indexed (index area).
*	Organization not recognized.

Example 1:

```
LS  -PN  ^Z00B00  START_UP.EC
```

List the attributes of the file START_UP.EC, contaned in the directory Z00B00. The following display is returned to the user-out file.

```
DIRECTORY: ^Z00B00

                                STARTING  NUMBER OF  RECORD
ENTRY NAME      TYPE            SECTOR    SECTORS     LENGTH
*************************************************************
START_UP.EC     S               35A       10          100
*************************************************************
```

Example 2:

```
LS  -DTL  -PN  ^Z00B00  XXXXXX
```

List in detail the file type and attributes of the file XXXXXX contained in the directory Z00B00. The following is written to the user-out file:

```
                                STARTING  NUMBER OF  RECORD
ENTRY NAME      TYPE            SECTOR    SECTORS     LENGTH
*************************************************************
CRDFILE01       S               24        A           100
                CURRENT         ALLOCATION          MAX
                ALLOCATION      INCREMENT           ALLOCATION
                SIZE                                SIZE

                ****************  ****************  ******************
                4               A                   0

                CI SIZE         LOGICAL             FREE SPACE
                                REC SIZE            PER CI
                ****************  ****************  ******************
                100             100                 0

                LAST CI LOADED
                **************************************
                4
*************************************************************************************
```

Note:

A Maximum Allocation Size of 0 means the file length is unlimited (i.e. physical limit of the media).

LAST CI LOADED refers to the logical end of data.

Free Space Per CI refers to the percentage of a data control interval (CI) which was initially left free for later file expansion (i.e. no data was initially loaded into it).

LIST SEARCH RULES

Command Name: LSR

Display the search rules currently defined for the issuing task group.

FORMAT:

LSR

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The LIST SEARCH RULES command writes to the user-out file the full pathnames of the directories used by the loader in its search for bound units.

The search rules define three directory pathnames and the sequence in which they are used during a search. The first of these is the issuing task group's working directory, if specified. The second is the system directory LIB1. The third is the system directory LIB2. The pathnames associated with LIB1 and LIB2 can be changed by the CHANGE SYSTEM DIRECTORY operator command. The pathnames returned by the LSR command always reflect the current directory pathnames.

Example:

Assume that the issuing task group's initial working directory is ^SYSVOL, that the pathname value for LIB1 and LIB2 is ^SYSVOL>SYSLIB1, and that no CWD or CSD commands have been issued. The LSR command returns:

```
^SYSVOL
^SYSVOL>SYSLIB1
^SYSVOL>SYSLIB1
```

Assume now that a CSD NEW_DIR -LIB2 command has been executed at some point prior to the issuing of the LSR command. The LSR command now returns

```
^SYSVOL
^SYSVOL>SYSLIB1
^SYSVOL>NEW_DIR
```

LIST WORKING DIRECTORY

Command Name: LWD

List the full pathname of the current working directory.

FORMAT:

LWD

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The LIST WORKING DIRECTORY command is used to obtain the full pathname of the working directory currently being used by the issuing task group. It is useful to be able to determine the identity of the working directory after having made several changes of working directories through the use of CHANGE WORKING DIRECTORY commands. The LWD command causes the full pathname of the working directory to be written to the user-out file in the form

^vol_id>dir$_1$ . . .

The ellipsis indicates that one or more subordinate levels may be included in the pathname of the working directory, depending on the nature of previously-issued CWD commands. Also, again depending on previous CWD commands, the person and/or project entries may not be included in the path.

Example:

Assume that a task group's initial working directory is ^SYSVOL>UDD>A1>JOE, as established at task group initiation.

A CWD EC_DIR command has been previously issued. The LWD command returns

^SYSVOL>UDD>A1>JOE>EC_DIR

If, starting with this working directory, a CWD > command is issued, a subsequent LWD command would return

^SYSVOL>UDD>A1>JOE

LOGIN

Command Name: L

The login command is used to gain access to the system. The login command is entered from any terminal not designated as a direct-login terminal or an abbreviated-login terminal. (To determine the type of terminal he is at, the user should contact the installation supervisor.) The login command causes a task group associated with the user's terminal to be spawned. Once he has access to the system, the user cannot again invoke login unless he first uses the BYE command or the task group is otherwise terminated.

FORMAT:

L [login_id] [destination_id] [ctl_arg]

ARGUMENT DESCRIPTION:

login_id

Establishes the identity of the user who is attempting to gain access to the system. Provides the user identification for the spawned task group. The login_id argument consists of from one to three fields having the following meanings:

person

person.account

person.account.mode

person — Name of person who may access system; can be from 1 through 12 characters. (For example, WDSMITH could be the value for the person field.)

account — Name of an account under which the user is to work; can be from 1 through 12 characters. (For example, JSINVENTORY could be used as the value for the account field.)

mode — Provides a further identification of the user; can be from 1 through 3 characters. (For example, VER could be used as the value for this field.)

[destination_id]

Optional argument that permits the user to login as a secondary user of an existing task group. (It is necessary that the running task group have previously issued a request for a secondary user terminal; the request for a secondary user terminal is entered by means of a Request Terminal macro vall; see the *GCOS6 System Service Macro Calls* manual for the format of the Request Terminal macro call.) To login as a secondary user of a user-created applications program, the user enters the value nn, where nn is the task group id of the task group in which the application is running. To login as a secondary user of task group $T (Terminal Concentrator), see the *Terminal Concentration Facility User's Guide.* When destination_id is specified, no control arguments can be selected. If the secondary login capability is not desired, then destination_id is omitted.

ctl_arg

One or more of the following control arguments can be selected:

{-PO path [id]
-PO * id }

Used to override the default lead task and group id/pool id specifications for the task group spawned as a result of this login procedure.

path

Pathname of the bound unit to be executed as the lead task of the spawned task group. If this argument is omitted, the lead task is the command processor.

id

Group id/pool id of the spawned task. The group id and the pool id are represented by the same 2-character value. If this argument is not specified, the group id is a

2-character value whose left (first) character was specified when the Listner component was activated (the Listner is described in the *Operator's Guide*) and whose right (second) character is the next unused character in sequence 0 through 9 and A through Z, as selected by the system. The First character of the id cannot be $.

-HD path
Used to specify the working directory for the task group spawned as a result of the login procedure.

path
Pathname of the initial working directory for the spawned task group. If this argument is omitted, the working directory pathname is null.

-LRN n
Used to override the default maximum logical resource number (LRN) value for the task group spawned as a result of this login procedure.

n
Maximum LRN value to be used for the spawned task group. (The maximum possible LRN value is 252.) If this argument is omitted, the maximum LRN value is the highest value in the system group.

-LFN n
Used to override the default logical file number (LFN) value for the task group spawned as a result of the login procedure.

n
Maximum LFN value to be used for the spawned task group. (The maximum possible LFN value is 255.) If this argument is omitted, the maximum LFN value is 15.

-ARG arg arg . . . arg
Used to pass additional arguments to the lead task of the task group spawned as a result of this login procedure. These additional arguments are passed to the spawned task in an extention of the task request block, and are substituted for parameters in the command-in file. If used, the -ARG control arguments must appear last. Refer to Appendix A for an explanation of the use of additional arguments.

The arguments will appear in the task request block extension in the following manner:

- Argument 1 will always be null.
- If the lead task is the command processor, argument 2 will be the pathname of the user-in file (i.e., >SPD>dev_name) and arguments 3 through n will be the arguments following -ARG.
- If the lead task is not the ECL processor, arguments 2 through n will be those arguments following -ARG.

FUNCTION DESCRIPTION

The login procedure allows the user to gain access to the system from a terminal. If the user is at a direct-login terminal, he does not use this command; rather, he is logged on automatically when his terminal is connected to the system. If the user is at a terminal that allows login abbreviation, he simply types the single character abbreviation. Both direct login and login by abbreviation are installation-specific features. To determine the type of terminal he is at, the user should contact the installation supervisor.

The user's terminal can be a noncommunications terminal (MDC-connected) or a communications terminal (MLCP-connected). At login initialization, the typeouts that appear at the terminal vary according to whether it is a communications or noncommunications terminal. See Appendix A for further information.

If the user enters an incorrect argument, the following message is displayed on the terminal:

LOGIN: 39xx (optional message from error message library)
RETRY — LOGIN INCORRECT

The user should retype the entire command or type BYE to terminate the session.

If a fatal error relating only to the user's terminal occurs, the following message is displayed:

```
LOGIN: 39xx (optional message from error message library)
HANGUP — FATAL ERROR
```

The user's terminal will be disconnected.

If no error conditions are encountered, the login procedure spawns the group to be associated with the user's terminal. The group id (and pool id) of the spawned task is determined by the value specified (for defaulted to) in the -PO path id argument. If the system-generated group id is in use, or a corresponding memory pool does not exist, another group id/pool id pair will be generated and another group will be spawned with this group id.

The spawned task group receives all but three of its arguments from the login line. Those arguments not received from the login line are base level, user-in, and user-out. The base level argument is received from the terminal login characteristics file (described in the *Operator's Guide*). The user-in and user-out arguments are generated as >SPD>dev_name, where dev_name is a name known to the file manager for the terminal for which the login process is spawning the group.

If the login is not a direct login, memory used for the login will be returned to the memory pool when no longer needed.

The following examples illustrate use of the login command.

Example 1:

```
L  JONESAC.PAYROLL2
```

The user logs in from a terminal using a full login command. The login id, which will be used in the identification of the spawned task group, is JONESAC (person) PAYROLL2 (account). The spawned task group will take all system defaults.

Example 2:

```
L  SMITH2  -PO  PROG1  -LFN  117
```

The user logs in from a terminal using a full login command. This command overrides two system defaults for the spawned task group. The lead task is to be the user-written procedure PROG1 and not the command processor. The maximum LFN value is to be 117 instead of 15.

Example 3:

```
F
```

The user logs in from a terminal allowing login by abbreviation. The login argument set identified by the abbreviation F will be processed by the login procedure and a task group will be spawned for the terminal.

MACRO PREPROCESSOR

Command Name: MACROP

Expand assembly language macro calls and %INCLUDE statements into assembly language source statements, applying the indicated options.

FORMAT:

MACROP path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the file which contains the unexpanded source statements. The file's .P suffix must not be included in the path argument; it is appended by the macro preprocessor prior to searching for the source unit.

[ctl_arg]
Zero, one or more control arguments chosen from the following list:

{-INCLUDE_CONTROLS
-IC}

Instructs the Macro Preprocessor to incorporate as comment statements in the expanded source output all macro control statements and inline macro definitions.

If a statement is in error, the Macro Preprocessor flags the statement, converts it to a comment statement, and writes it to the expanded source file.

If this argument is omitted, these comment statements are omitted.

{-SIZE nn
-SZ nn}

Designates the maximum number of 1024-word blocks of memory that the Macro Preprocessor can use for work space; nn must be a decimal representation from 01 through 99. If this argument is not specified, two blocks (2048 words) of memory are used.

{-MACRO CALLS
-MC}

Causes all macro call statements as comments in the output.

FUNCTION DESCRIPTION:

The MACROP command is used to invoke the Macro Preprocessor component.

The input path argument can assume any of the acceptable forms of a pathname; a relative name indicates that a source program unit residing in the working directory is to be expanded. Whereever the source program unit exists, it must have the .P suffix. However, the path argument in the command cannot have the .P suffix. The output is always directed to file path.A in the working directory.

The MACRO Preprocessor always writes to the error-out file the number of errors encountered. Specification of the -IC argument enables the user to examine the statements that generated errors.

A full description of the operation and use of the Macro Preprocessor is contained in the *Assembly Language* manual.

MERGE FILES

Command Name: MERGE

Merge the records of up to six files.

FORMAT:

MERGE [ctl_arg]

ARGUMENT DESCRIPTION:

[ctl_arg]
One or more control arguments chosen from the following list.

-IN path
Specifies the name of the file containing the merge descriptors for this merge. If not specified, the user-in file is assumed to contain the merge descriptors.

-PD
Indicates that a listing of the merge descriptors is to be produced on the user-out file. (Only the first 71 characters of the line will be displayed.) If not specified, no list is issued.

-DL
When duplicate records are encountered, delete all occurrences of the records except the first.

FUNCTION DESCRIPTION:

The MERGE command provides the capability of merging the records of up to six files according to specifications supplied in a merge descriptor file.

A complete description of the operation and use of the merge component is contained in the *Sort/Merge* manual.

MESSAGE [MOD 400 ONLY]

Command Name: MSG

Send a message from a user command device to the operator terminal.

FORMAT:

MSG message

ARGUMENT DESCRIPTION:

message
The message to be sent. If the message contains embedded blanks, it must be enclosed in double quotes (") or apostrophes (').

FUNCTION DESCRIPTION:

The MSG command is used whenever it is necessary for a task group to convey some item of information or a request for operator action to the system operator. The source of the message is whatever file or device is designated as command input for the sending task group at the time the message is sent; the message is displayed on the operator terminal.

Example:

MSG "PLEASE ABORT BATCH REQUEST"

Send a message to the operator requesting an abort of the current batch request. The operator responds by entering an ABR operator command.

MODIFY EXTERNAL SWITCHES

Command Name: MSW

Modify selected external switches associated with the issuing task group.

FORMAT:

MSW ctl_arg

ARGUMENT DESCRIPTION:

ctl_arg

One or more control arguments chosen from the following list:

-ON $S_i[S_i]$ . . .

Set the external switch indicated by S_i ON. Each S_i is a hexadecimal digit from 0 through F.

-OFF Si[Si] . . .

Set the external switch indicated by S_i OFF. Each S_i is a hexadecimal digit from 0 through F.

-ALL v

Set all switches to the value v. The value v can be either ON or OFF.

FUNCTIONAL DESCRIPTION:

The MODIFY EXTERNAL SWITCHES command enables the issuing task group to modify the external switches by which it can control its execution. An external switch can be thought of as a hardware switch on a control panel, which can be set on or off manually by an operator. There is a separate switch word associated with each task group created, giving each group the capability of addressing 16 switches. A user program can contain instructions or statements which interrogate the settings of one or more of these switches, and can use these settings to control the execution logic of the program.

Example:

MSW -ON 25 -OFF 7B

In the issuing task group, external switch numbers 2 and 5 are to be set ON, and external switch numbers 7 and B are to be set OFF.

MODIFY FILE

Command Name: MF

Modify the attributes of the specified file.

FORMAT:

MF path ctl_arg

ARGUMENT DESCRIPTION:

path
The pathname of a file whose attributes are to be changed.

ctl_arg
One or more control arguments chosen from the following list. At least one is required; there are no defaults.

{-SHARE / -SHR}
Specifies that the named file is to be made accessible to the batch task group.

{-NONSHARE / -NS}
Specifies that the named file is to be made inaccessible to the batch task group.

{-READ / -RD}
Specifies that no users are given permission to write to the named file; only reading is permitted.

{-WRITE / -WR}
Specifies that users are permitted access to the named file in the output, update, or extend mode.

Note:
The arguments within the argument pairs -SHARE and -NONSHARE, and -READ and -WRITE, are mutually exclusive.

The MODIFY FILE command allows the accessibility and permission attributes of a file to be modified. When a file is first created (refer to the CREATE FILE command in this section), it is accessible to both online and batch task groups. It can also be read from and written to by any task. Its initial attributes are thus SHARE/WRITE.

If a file is made inaccessible to the batch task group (through the use of the -NS control argument), no access of any kind by the batch task group is permitted. Furthermore, directories can be given the -NS attribute; in this case the directory and all subdirectories and files contained within it are inaccessible to the batch task group.

Read protection can also be given a file by the use of the -RD control argument. This argument makes the file a read-only file, preventing any task groups, online or batch, from writing to the file. It can still be read by online tasks and, unless the -NS argument has also been specified, by batch tasks as well.

Attributes assigned to nondisk files by this command remain in effect only for the current initialisation of the system. If the system is reinitialized, attributes for these files revert to SHARE/WRITE. The MF command can be issued only from an online task group.

If the file to be modified is currently reserved by a task group, the request is denied.
Example:

```
MF  >UDD>PROJ1>USERA>FILE01  -NS
```

A file is to be made inaccessible to the batch task group. The read/write protection remains unchanged.

NEW PROCESS

Command Name: NEW_PROC

Abort the current task group request and restart the task group using the same arguments as specified in the original group request.

FORMAT:

NEW_PROC

ARGUMENT DESCRIPTION:

Arguments are neither required nor permitted with this command.

FUNCTION DESCRIPTION:

The NEW_PROC command suspends all tasks of this task group. It removes all task structures (except the lead task — the command processor), returns all memory to the task group's memory pool, and closes and releases all files. In effect, the task group is deleted and restarted with the original arguments.

PATCH

Command Name: PATCH

Patch an object or image text file.

FORMAT:

PATCH path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the object or image text file to be patched. An object text pathname must end with the .O; No suffix is used for an image text file unless one was assigned when the image text unit was linked.

[ctl_arg]
One of the following control arguments can be specified:

-IN path
The pathname of the file which contains the patch directives. If not specified, the directives are read from the current user-in file.

{-PROMPT / -PT}
Can be specified only if directives are read from the user-in file. Causes the PATCH processor to issue a P? prompting sequence each time it is ready to receive an input line. If this argument is omitted, no prompting sequence is printed.

FUNCTION DESCRIPTION:

The PATCH command permits the selective modification of an object or image (bound unit) text file, in accordance with directives submitted to the PATCH processor. A complete description of the directives and operation of the PATCH command, as well as several examples of its use, appear in the *Program Execution and Checkout* manual.

Patching a Monitor component while it is executing is not recommended. The results of such patching are unspecified.

PRINT

Command Name: PR

Print the contents of the indicated file.[2]

FORMAT:

PR path [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the file whose contents are to be printed.

[ctl arg]
One or more control arguments chosen from the following list:

{-LIMIT nn / -LI nn}
Specifies the number of records to be printed if end of file is not encountered before the value of nn is satisfied. If not specified, all records in the file are printed.

{-COPIES n / -CP n}
Specifies the number of copies to be printed; i.e., the number of times the file is to be printed for this invocation. Default is 1.

{-SPACE n / -SP n}
This argument indicates that the file is not a true print file with print control characters in its records. Each record is printed on one or more print lines. The value of n specifies the line spacing between records, and can be either 1 or 2. 1 specifies single spacing (no blank line). 2 specifies double spacing (one blank line). The default value for n is 1. If this parameter is not specified, the first record byte is treated as a printer control character, i.e.; the file is assumed to be a print file. See the control byte description for the printer driver in the *System Service Macro Calls* manual.

{-FORTRAN / -FTN}
The print file was created by a FORTRAN object program and has print control characters of the FORTRAN type.

{-FROM nn / -FM nn}
Indicates that the first nn records of the file are to be skipped before printing begins. If not specified, printing starts at beginning of file.

{-LINE_LEN nn / -LL nn}
Specifies the number of characters to be printed per line. If a longer line is read from the file, it is folded at the indicated print position. If not specified, the value of nn is 68.

{-RELEASE / -RL}
Specifies that, at the completion of printing, the file is to be released.

FUNCTION DESCRIPTION:

The PRINT command is used to write to the current user-out file the contents of a file formatted according to the system print file conventions. Such a file contains, for each record to

[2] See also Deferred Print command earlier in this section.

be printed, a printer forms control byte. This byte is in the first character position of each record, and serves to control the line spacing associated with each print line, as well as head-of-form spacing.

Print files written by the various language processors are suffixed with a .L unless otherwise directed by the processor's -COUT control argument. This suffix must be included in the pathname specified by the path argument when the PRINT command is used to print these types of files. These files always contain forms control bytes. User programs which write files destined to be printed using this command are responsible for supplying the appropriate forms control bytes in their output records. Files written by user programs are not required to be terminated with the .L suffix.

Print files written by FORTRAN object programs utilize a special set of forms control bytes. If the PRINT command includes the -FTN control argument, these bytes are translated into equivalent standard forms control actions before the line is printed. If the -FT control argument is not specified for these files, resulting form spacing will probably not reflect that which was intended by the programmer.

Theoretically, any file can be printed by using the PRINT command. However, since the first byte of each record is interpreted as a forms control indicator, the line spacing which results from the printing of a nonprint file is unspecified. The -SP control argument, in addition to specifying the spacing between records, also negates the interpretation of the first byte as a control byte. Each record is printed on as many single-spaced lines are are required and the line spacing between records then occurs as specified by the -SP argument. When the -SP argument is used, the first byte of each record appears in the print line.

The user can request the printing of only a part of a file by the appropriate combination of -FM and -LI control arguments, which define, respectively, the point in the file at which printing is to begin and the number of lines to be printed.

When the output of the PRINT command is directed to a high-speed printer, the use of the -LL control argument specifying the physical line length of the printer is recommended, since the length of an output record whose ultimate destination is such a device is likely to be longer than the default 68 characters. If the argument is not specified, each such line will be folded at the 68th character. If a line is folded, the continuation line starts with a /C.

When an entire file is being printed (no -FROM or -LIMIT argument), the first page of file data is automatically preceded by a header page containing the pathname and the current date. No header page precedes the file data of a partial file.

When end-of-file is encountered, an end-of-file message is printed on the page following the last page of file data.

Example 1:

```
PR  TABLIST  -CP  2  -FT
```

Two copies of the print file written by a FORTRAN program are to be printed. The file contains print lines less than 68 characters long; hence, the -LL argument is not required. The printed output is written on whatever device is currently associated with the user output file.

Example 2:

```
PR  COBPRINT  -LL  132
```

The print file from a program which writes 132-character print records is to be printed. If the current user output device is not a line printer, the PR command can be preceded by an FO (FILE OUT) command naming a line printer (LPTnn) as the output device.

READY OFF

Command Name: RDF

Suppress the 'ready' message printed at the completion of each command.

FORMAT:

RDF

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The READY OFF command suppresses the printing of a message issued by the system at the completion of execution of each command. The message informs the user that the system is prepared to accept another command.

The initial state of the ready function at the conclusion of task group initiation or the beginning of EC execution is OFF.

If the RDF command is issued from within an EC file when execution of the EC file is completed, the system reverts to the ON/OFF state which was in effect when the EC command was invoked.

The function must be requested with the command name shown; i.e., RDF.

READY ON

Command Name: RDN

Activate the printing of the ready message at the completion of each command.

FORMAT:

RDN

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The READY ON command activates the printing of a message, RDY:, issued by the system at the completion of execution of each command. The message informs the user that the command processor is prepared to accept another command.

The initial state of the ready function at the conclusion of task group initiation or the beginning of EC execution is OFF.

If the RDN command is issued from within an EC file, when execution of the EC file is completed the system reverts to the ON/OFF state which was in effect when the EC command was invoked.

The "RDY:" message invoked by the RDN command is transmitted to the user_out file, even if it is not a terminal. User_out is altered by the FILE OUT command. If RDN is in effect, it will not issue the "RDY:" message after & statements.

The function must be requested with the command name shown; i.e., RDN.

RELEASE

Command Name: RL

Release the space occupied by the named directory or file to the File System.

FORMAT:

RL [-FILE / -DIR] path . . .

ARGUMENT DESCRIPTION:

path
Specifies the pathnames of one or more file system entries to be released. Can be any valid form of pathname; can use the star convention (see Section 1).

-FILE
Indicates that the entries to be released are files. This is the default if neither -FILE nor -DIR is specified (see Example 2, below).

-DIR
Indicates that the entries to be released are directories.

FUNCTION DESCRIPTION:

The RELEASE command, when issued to release a file, removes all of the file's attributes from the directory within which it is immediately contained. All of the space which was allocated to the file is returned to the File System. If the file is open at the time the RL command is issued, the file is not released. If the file is reserved for use by another task group (i.e., another task group has issued a GET command specifying this pathname), the file is released after that task group has closed the file and issued a REMOVE command.

When used to release a directory, the RL command removes all of the directory's attributes from the immediately superior directory. All of the space which was allocated to the directory is returned to the File System. The directory to be released must be "empty"; i.e., it cannot contain entries representing subdirectories or files. If it is not empty, it is not deleted.

Example 1:

```
RL  -FILE  FILE01
```

The file, FILE01, in the working directory is released if it is not in use or reserved by another task.

Example 2:

```
RL  SUB_DIR1>FILE02
```

The file, FILE-2, in a directory SUB_DIR1, immediately subordinate to the working directory, is released if it is not in use or reserved by another task.

Example 3:

```
RL  -DIR  SUB_DIR1
```

The directory, SUB_DIR1 immediately subordinate to the working directory, is released, provided it is empty.

REMOVE FILE

Command Name: REMOVE

Cancel a previous file reservation.

FORMAT:

Format 1 (Pathname):

REMOVE path [ctl_arg]

Format 2 (Logical File Number):

REMOVE lfn [ctl_arg]

ARGUMENT DESCRIPTION:

path

The pathname of a file whose reservation is to be cancelled. The reservation of all logical file numbers (LFNs) associated with the pathname will be cancelled. This argument applies only to Format 1.

lfn

The logical file number (LFN) of the file whose reservation is to be cancelled. This argument applies only to Format 2.

[ctl_arg]

The following control argument can be specified.

-FORCE

Force the cancellation of all reservations associated with this pathname, even if the file is still open.

FUNCTION DESCRIPTION:

The REMOVE command cancels the reservation of a file effected by a previously issued GET command. If the file is open, the remove action will not take effect and an error will be displayed. Either a specific LFN or all LFNs associated with this pathname are to cancelled. The -FORCE argument forces the cancellation to occur even if the file is still open. Subsequent access to this file will return an "unknown or illegal LFN" error (0206).

Example 1:

REMOVE BOOKS>FILEB

The reservations of all LFNs associated with the file whose pathname is BOOKS>FILEB in the current working directory are cancelled. Cancellation does not occur if the file is open.

Example 2:

REMOVE 22 -FORCE

The reservation of the file associated with LFN 22 is cancelled. Cancellation occurs even if the file is still open.

RENAME FILE

Command Name: RENAME

Assign a new name to an existing file or directory.

FORMAT:

RENAME oldname newname

ARGUMENT DESCRIPTION:

oldname
The present pathname of the file or directory to be renamed. Can use starname convention. (See Section 1.)

newname
A simple name unique within the directory containing oldname. Can use equal name convention. (See Section 1).

FUNCTION DESCRIPTION:

The RENAME command is used to change the name of an existing file or directory.

The oldname parameter can be a simple, relative, or absolute pathname. The only requirement is that the specified file exist in the expressed or implied directory. If a simple name is given, the file must exist in the working directory. If a relative or absolute pathname is given, the file must exist in the directory derived from the given pathname.

Whatever directory is established by the oldname argument is the one in which the file will reside under its new name. The new file name must be one which does not already exist in that directory, in accordance with the requirement that, within a given directory, all file names must be unique.

Example 1:

Assume a working directory >UDD>BOOKS>SMITH, and that in this directory there is a file AB. The command

RENAME AB CD

changes the pathname of the affected file from

>UDD>BOOKS>SMITH>AB to >UDD>BOOKS>SMITH>CD.

Example 2:

Assume that within the working directory in example 1 is a subdirectory CHANGES, which contains a file AB_CHANGES. The command

RENAME CHANGES>AB_CHANGES CD_CHANGES

implies the directory >UDD>BOOKS>SMITH>CHANGES, since the oldname argument is in the form of a relative pathname. The pathname of the file within this directory is changed to >UDD>BOOKS>SMITH>CHANGES>CD_CHANGES.

RESET MAP

Command Name: RS

Reconstruct the bit map on a disk volume or list the number of unused logical sectors available for allocation on the volume.

FORMAT:

RS path [ctl_arg]

ARGUMENT DESCRIPTION:

path
Specifies the name of the volume whose unused sectors are to be listed or whose bit map is to be reset. The form of path is >SPD>dev_name>[>vol_id]. If vol_id is present, the volume name is verified.

ctl_arg
The only control argument is the following:

-AVAIL
List the number of available (unused) sectors on the named volume. When -AVAIL is specified, the bit map is *not* reset, only this list operation is performed.

FUNCTION DESCRIPTION:

The RESET MAP command with the -AVAIL argument lists the number of logical sectors that the volume bit map indicates as being available for allocation. Using the logical sector size specified in the volume label, the number is converted to physical sectors. The following message is issued:

VOLUME vol_id HAS AVAILABLE sss LOGICAL SECTORS = pppp PHYSICAL SECTORS

The bit map is not modified.

The file system maintains a map which represents the locations of available blocks of disk storage on a volume. The map for each volume is stored on the volume itself. It reflects information contained in directories and subdirectories relating to the locations and extents of the files residing on the volume. If necessary, the RS command can also be used to construct a new available-sector map. It interrogates all directories and subdirectories on a volume for file location and extent information and, using this information, constructs a new map.

The message (12121D), A LOGICAL SECTOR OF THE FILE 'VOLID' WAS PREVIOUSLY ALLOCATED, will be issued by the RESET MAP program when the volume major directory of a cartridge disk is in its default location. This message should be ignored.

RESTORE
[MOD 400 ONLY]

RESTORE

Command Name: RESTORE

Restore the specified files previously saved by the SAVE utility.

FORMAT:

RESTORE save-file name [starting directory]

ARGUMENT DESCRIPTION:

save-file name
Specifies the pathname of the file to be restored (i.e. the output file from the previous SAVE).

[starting directory]
Specifies the pathname of the directory on the output volume where the restore is to begin. This directory *must* exist. It is not created by the RESTORE utility. If no directory name is specified, the RESTORE utility uses the pathname stored in the save-file by the SAVE utility.

FUNCTION DESCRIPTION:

The RESTORE command restores a file or directory structure saved by the SAVE command. If a file or directory exists by the same name as the one being restored, it is overwritten with the ACL remaining intact. Otherwise, the file is created with ACL data from the saved file. Any unsaved files or directories in the specified directory (or in subordinate directories during the restore) are preserved since the restored data is appended. When the RESTORE command is in effect, the volume is reserved with write exclusive access.

RPG

Command Name: RPG

Compile the RPG source program unit represented by the indicated file name, applying the specified compiler options.

FORMAT:

RPG path [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the source unit file to be compiled (without the .R suffix). ".R" is automatically appended before searching for the source unit-file.

[ctl_arg]
None or any number of the following control arguments can be entered, in any order.

{-COUT / -C} out_path

Listing will be written to the file out_path; a suffix is not automatically appended to the file name; out_path may specify the line printer. If this argument is omitted, the listing will be written to the file source.L in the working directory (source is the simple name of the source unit file to be compiled).

{-LIST_OBJ / -LO}

List data map and object text, in addition to the source text, the diagnostics and the Linker command file listings.

{-NO_LIST / -NL}

Suppress all listings.

{-NO_OBJ / -NO}

Suppress object unit output.

Default: Object units are produced in the working directory as a number of files, each having the .O suffix and a compiler-generated name. If a particular .O file already exists in the working directory, it is overwritten by the output of the current compilation.

{-SIZE nn / -SZ nn}

nn designates the maximum number of 1024-word blocks of additional memory that the RPG Compiler may use for tables; nn must be from 04 to 28.

Default: 03

FUNCTION DESCRIPTION:

The RPG command is used to invoke the RPG Compiler component.

The path argument can assume any of the acceptable forms of a pathname, although normally it may be a simple name, indicating that a source program unit which resides in the current working directory is to be compiled. Wherever the file exists, it must have a .R suffix, indicating that it is an RPG language source unit. The path argument must be given *without* the .R suffix; the compiler appends the suffix prior to searching the directory for the source unit.

If the -COUT control argument is not specified, the listings are written to a file created by the compiler in the working directory, having a file name of the form source.L. If a different file is specified by using the -COUT argument, the listings are written to the file whose pathname is out_path. In either case, the file can later be output to a line printer, by using the PRINT utility command. The compiler does not append a .L suffix.

If a file of the form source (our out_path) already exists in the working directory, it is overlaid by the output of the current compilation.

Notes:

1. Either -LO or -NL can be specified, but not both. If neither is specified, the compiler produces a listing of the source text, the diagnostics, and the Linker command file.
2. The RPG compiler always writes the number of diagnostics produced on the error-out file.

Example:

```
RPG  RGPROG  -COUT  RGPROG_LST  -LIST_OBJ
```

An RPG source program, RGPROG.R, located in the working directory, is to be compiled. Listings are to contain source statements, error diagnostics, data map, object code and Linker commands and are to be written to a file named RGPROG_LIST, in the working directory.

SAVE

Command Name: SAVE

Save the specified disk directories and/or files.

FORMAT:

SAVE file/directory name output file [-LEV n]

ARGUMENT DESCRIPTION:

file/directory name
Specifies the pathname of the directory where the save is to begin, or specifies the pathname of the file(s) to be saved.

output file
Specifies a device, a tape file or a disk file where the data being saved is to be recorded. If the specified file already exists, it will be opened in renew-mode, replacing any previously existing data in the file. If the disk file does not exist, it will be created.

[-LEV n]
For directory saves, "n" specifies how many levels of directories are to be included in the save-file. This parameter is ignored for file saves. If omitted, all subdirectories are included in the save (default). A value of 1 for "n" directs the utility to save only the file entries in the specified directory, which excludes all subordinate directories and files.

FUNCTION DESCRIPTION:

The SAVE utility provides a backup facility and volume reorganization capabilities. Its function is complemented by the RESTORE utility described in this manual.

SET ACCESS CONTROL LIST

Command Name: SET_ACL or SA

Manipulate the access control list (ACL) of a file or directory by adding new entries or changing the access mode of existing entries.

FORMAT:

SET_ACL path access_mode user_id [ctl_arg]

ARGUMENT DESCRIPTION:

path
Pathname of the file or directory containing the ACL. If this argument is -WD, the working directory is used.

access_mode
Specifies an access mode for the directories or files. (Path specifies whether a directory or file is being operated upon.)

Any or all of the following values may be specified for files:

R — Read access

E — Execute access

W — Write access

Any or all of the following values may be specified for directories:

L — List access

M — Modify access

C — Create access

One of the following values may be specified for files or directories; if used, it must be the only entry:

N — Null access

Δ — Null access

user_id
An access control name having the following format:

person account mode

Existing ACL entries that have matching access control names receive the access mode specified by the <access_mode> argument.

If no matching entry is found, the entry is added to the ACL, provided that each component of the access control name resolves to a literal value.

If the user_id argument is not specified, the current user id is employed, with the following format:

person.account.*

ctl_arg
The following optional control arguments can be chosen:

{-DIR
-DIRECTORY}

Specifies that only directory values are to be allowed for the access_mode argument; if this argument is not specified, the first access_mode value specified sets the default.

-FILE
Specifies that only file values are to be allowed for the access_mode argument; if this argument is not specified, the first access_mode value sets the default.

FUNCTION DESCRIPTION:

Access control lists (ACL's) and common access control lists (CACL's) are an optional system feature that afford variable types of protection for mass storage directories and files.

ACL's can be set for any or all directories and files on mass storage volumes. An ACL for a *directory* indicates which users can create, modify, and/or list directories or files immediately subordinate to this directory. An ACL for a *file* indicates which users can read, write, and/or execute (if it is an executable entity) the file.

CACL's can be set at any or all directories on mass storage volumes. They are classified in two types: directory CACL's and file CACL's. A directory CACL indicates access rights for *all directories* immediately subordinate to the directory at which the directory CACL is set; a directory CACL is thus equivalent to an identical ACL being set for *each* of the immediately subordinate directories. A file CACL indicates access rights for *all files* immediately subordinate to the directory at which the file CACL is set; a file CACL is thus equivalent to an identical ACL being set for *each* of the immediately subordinate files.

The access rights defined by a directory CACL entry or by a file CACL entry can be overridden, on an individual basis, by ACL entries set on specific, immediately subordinate directories and files.

ACL's are set for directories and files by means of SET_ACL commands. ACL's are listed by LIST_ACL commands and are deleted by DELETE_ACL commands. CACL's are set at directories by SET_CACL commands. CACL's are listed by LIST_CACL commands and are deleted by DELETE_CACL commands.

It is important to realize that although ACL's and CACL's provide a *static* description of which users may gain access to specific directories and files and what type(s) of access they have, ACL's and CACL's do *not,* of themselves, *guarantee* a user access to a directory or file *at all times. Concurrency* constraints may *at some times* prevent a user from gaining access to a directory or file to which he is otherwise entitled to access by virtue of an ACL or CACL. (For instance, if user A has obtained access to a given file with *exclusive* concurrency control, other users cannot gain concurrent access to this file even though existing ACL or file CACL otherwise would permit them access to this file.)

Once established by a SET_ACL command, an ACL for a given directory or file consists of one or more entries, each of which contains a user_id and an access privilege. The same is true of a directory CACL and a file CACL established at a given directory by a SET_CACL command.

The user_id in an ACL or CACL entry consists of the three elements shown below:

person.account.mode

Note the correspondence between the elements of this user_id and the identity established for a user as he logs in to the system. The significance of this correspondence is that, in an environment that employs access control, a user's access to directories and files is directly related to his person_id.account.mode identity as established as he logs in to the system.

An important feature of the user_id in an ACL or CACL entry is that any or all elements of the user_id may be expressed by an asterisk. The asterisk is equivalent to "any" and thus permits varying degrees of generality or comprehensiveness for the access privilege specified in the same ACL or CACL entry. The following examples illustrate this point:

1. A user_id expressed as SMITH.ACCT1.* applies the accompanying access privilege to SMITH under ACCT1 in ANY mode.
2. A user_id expressed as *.ACCT2.* applies the accompanying access privilege to ANY user under ACCT2 in ANY mode.
3. A user_id expressed as HIRSCH.*.* applies the accompanying access privilege to HIRSCH under ANY account and in ANY mode.
4. A user_id expressed as *.*.* applies the accompanying access privilege to ANY user under ANY account in ANY mode.
5. All other combinations are also legal.

As described later in this appendix under "Access Rights Checking," a hierarchical scheme — relative to how "explicitly" a user_id is expressed — governs whether the access privilege specified in an ACL entry will be overridden by the access privilege specified in a related directory CACL or file CACL entry.

The access privilege in an ACL entry for a file, and in a file CACL, can be set to one, two, or three of the elements shown below.[3] Permissible combinations are R, E, RE, RW, and RWE.

R — The user or users designated in this entry can *read* the file or files affected.

W — The user or users designated in this entry can *write* the file or files effected. (Write access to a file requires read access as well.)

E — The user or users designated in this entry can *execut* the file of files affected (provided the file is an executable entity).

The access privilege in an ACL entry for a directory, and in a directory CACL, can be set to one, two, or three of the elements shown below. Permissible combinations are C, L, LM, LC, and LMC.

C — The user or users designated in this entry can *create* files and directories immediately subordinate to the directory or directories affected.

M — The user or users designated in this entry can *modify* files and directories immediately subordinate to the directory or directories affected. (Modify access to a directory requires list access as well.)

L — The user or users designated in this entry can *list* files and directories immediately subordinate to the directory or directories affected.

When checking access rights, the system first compares the user's login identity (person_id.account.mode) against the entries in the ACL (if any) of the target directory or file.

- If a direct match is found (possible only if all elements of an ACL entry's user_id are explicitly stated—i.e., no asterisks), the user's access privilege is established by that ACL entry.
- If a direct match is *not* found, the system searches the ACL entries to find the one of the highest priority that *includes* the user's login identity. The priorities are shown below in decreasing order of priority.

  highest — person.account.mode
  person.account.*
  person.*.mode
  person.*.*
  *.account.mode
  .account.
  ..mode
  lowest — *.*.*

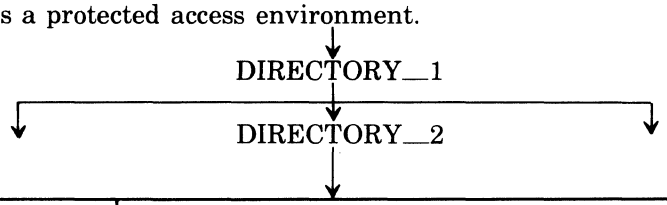
  When the system finds the highest priority ACL entry that *includes* the user's login identity (e.g., the user's login identity is ROTH.ACCT6.INT and the system finds an ACL entry whose user_id is *.ACCT6.*), the system searches the related directory CACL or file CACL (if any) trying to find a CACL entry of *higher* priority that includes the user's login identity. If the system *does find* such an entry in the related directory CACL or file CACL, the user's access privilege to the target directory or file is as established therein. If no directory CACL or file CACL exists or if one exists but it does not contain an entry of higher priority that includes the user's login identity, the user's access privilege is as established in the target directory's or file's highest priority ACL entry that includes the user's login identity.
- If the target directory or file contains *no* ACL entry that includes the user's login identity the directory CACL or file CACL (if any) is searched for the highest priority entry that

[3] Alternatively, the access privilege can be set to N (no access). If N is specified, it must be the *only* access privilege element.

includes the user's login identity. The user's access privilege to the target directory or file will be as established therein. (if the user's login identity is not included in *either* an ACL entry for the target directory or file *or* in an entry in a related directory CACL or file CACL, he has no access privilege in a protected access environment.)

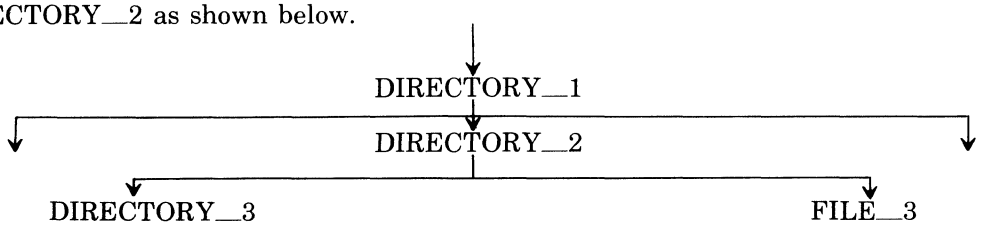
The following general guidelines and examples should prove helpful to your understanding of some of the basic characteristics of ACL's and CACL's.

For the purpose of the general guidelines, first consider a directory structure as depicted below. Assume that this is a protected access environment.



To Perform This Action at DIRECTORY_2	User Requires This Type of Access Privilege
create an immediately subordinate directory or file	C (create) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1
list any immediately subordinate directory or file	L (list) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1
release an immediately subordinate directory or file	M (modify) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1

Now assume that a DIRECTORY_3 and FILE_3 have been created immediately subordinate to DIRECTORY_2 as shown below.



To Perform This Action at DIRECTORY_3 or FILE_3	User[a] Requires This Type of Access Privilege
set-ACL	M (modify) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1
list-ACL	L (list) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1
delete-ACL	M (modify) access in an ACL for DIRECTORY_2 or in a directory CACL at DIRECTORY_1
set-common-ACL[b]	M (modify) access in an ACL for DIRECTORY_3 or in a directory CACL at DIRECTORY_2
list-common-ACL[b]	L (list) access in an ACL for DIRECTORY_3 or in a directory CACL at DIRECTORY_2
delete-common-ACL[b]	M (modify) access in an ACL for DIRECTORY_3 or in a directory CACL at DIRECTORY_2

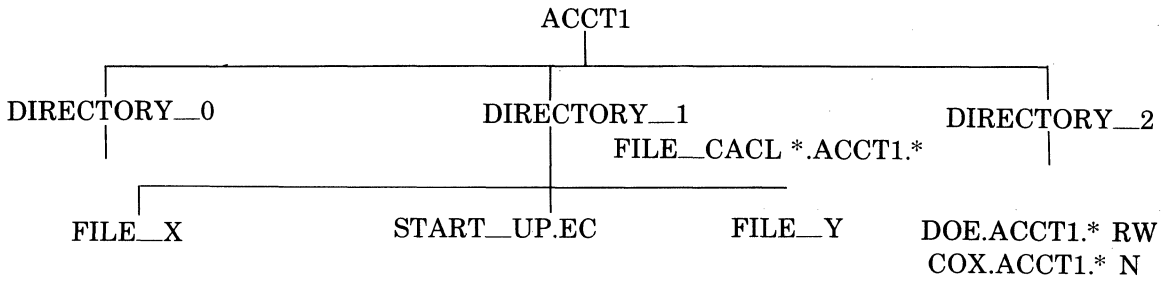
[a] This user need not necessarily be the one who created DIRECTORY_3 and/or FILE_3.

[b] Only at DIRECTORY_3.

The following examples illustrate a hypothetical case in which an account administrator has created several files and directories and then proceeds to set access privilege for account users. The examples illustrate a number of features of ACL's and CACL's but they are by no means comprehensive.

A number of other approaches to setting access privilege might be used.

Example 1:



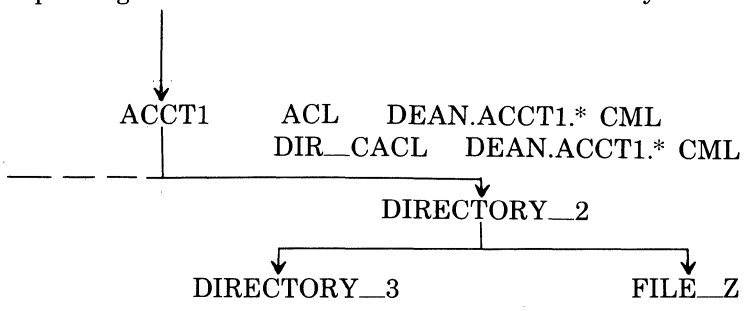
This example shows that the account administrator of ACCT1 has done the following:

1. He has created a file CACL entry at DIRECTORY_1 allowing all users of ACCT1 (in any mode) to read all files immediately subordinate to DIRECTORY_1. (Exceptions are noted below.)
2. He has set an ACL entry at FILE_Y allowing DOE to *read and write* the file. Thus DOE's access privilege to FILE_Y *exceeds* that of other users of ACCT1.
3. He has set an ACL entry at FILE_Y denying COX any access to the file. Thus COX's access privilege to FILE_Y is *less than* that of other users of ACCT1.

Both DOE and COX, like other users of ACCT1, have read access privilege to FILE_X and to START_UP.EC.

Note that although there is no ACL set directly at FILE_X or at START_UP.EC, no user except those in ACCT1 has access privilege to them because of the file CACL entry set at DIRECTORY_1.

Example 2:



This example shows that although DEAN under ACCT1 (in any mode) is the only user who can create DIRECTORY_3 and FILE_Z by virtue of the directory CACL entry at ACCT1), once they have been created, any user could create directories and files subordinate to DIRECTORY_3 and any user could set directory CACL or file CACL at DIRECTORY_3. These capabilities are available to all users because no directory CACL has been set at DIRECTORY_2. (This may be perfectly acceptable.)

If DEAN were to set a directory CACL entry for himself at DIRECTORY_2 *before* creating DIRECTORY_3 or FILE_Z, he could deny the above mentioned capabilities to all other users.

In any case, because of his access privilege at a higher level, DEAN could, by setting appropriate access privilege for himself, "undo" anything undesirable that a user had done at or below DIRECTORY_3.

SET AUTODIAL TELEPHONE NUMBER

Command Name: SDL

Insert the specified telephone number into the first entry in the autodial telephone number list for the specified line. This telephone number will be used first when the autodial facility attempts to establish a connection on the (switched circuit) line.

FORMAT:

SDL { channel / file } phone_number

ARGUMENT DESCRIPTION:

channel
Four hexadecimal digits that define the 10-bit (left-justified) channel number of the line whose telephone number list is to be altered.

file
Volume-level access pathname (e.g., >SPD>file_name) of the line (device) whose telephone number list is to be altered.

phone_number
The telephone number to be inserted in the first entry of the autodial telephone number list for the line. The value for phone_number is an ASCII string of 1 through 16 characters chosen from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, and *.

FUNCTION DESCRIPTION:

During system building, the user can specify that the communications autodial facility be applied to one or more communications lines. For each line that is to employ autodialing, the user constructs a list of telephone numbers. The first entry in this list is left empty by the system. The other entries are filled in according to the user's specifications.

The SDL command allows the user to dynamically insert a telephone number into the first entry in the list for a particular line. When the autodial handler is invoked, this telephone number will be dialed first in the attempt to establish a connection with the terminal(s) on the line. If no successful connection is established, the next entry (telephone number) in the list is dialed, and so on until a successful connection is made or every number in the list has been dialed. (Each telephone number is dialed 3 times at 40-second intervals.)

Example:

```
SDL   >SPD>TTY1   1-617-667-3111
```

The telephone number 1-617-667-3111 is inserted in the first entry in the autodial telephone number list used in dialing terminal TTY1.

SET COMMON ACCESS CONTROL LIST

Command Name {SCA | SET_CACL}

SCA

Manipulate the file or directory common access control list (CACL) of a directory by adding a new entry or changing the access mode of existing entries.

FORMAT:

{SCA | SET_CACL} path access_mode user_id [ctl_arg]

ARGUMENT DESCRIPTION:

path

Pathname of the directory containing the CACL. If this argument is -WD, the working directory is used.

access_mode

Specifies an access mode valid for directories or files (depending on whether a file or directory CACL is being added or changed).

Any or all of the following values may be specified for files:

R — Read access

E — Execute access

W — Write access

Any or all of the following values may be specified for directories:

L — List access

M — Modify access

C — Create access

One of the following values may be specified for files or directories; when the value is used, it must be the only entry.

N — Null access

"Δ" — Null access

user_id

An access control name having the following format:

person.account.mode

Existing CACL entries that have matching access control names receive the access mode specified by the access_mode argument.

If no matching entry is found, the entry is added to the CACL, provided that each component of the access control name resolves to a literal value.

If the user_id argument is not specified, the current user id is employed, with the following format:

person.account.*

[ctl_arg]

The following optional control arguments can be chosen:

{-DIR | -DIRECTORY}

Specifies that only directory values are to be allowed for the access mode argument. If this argument is unspecified, the first access_mode value specified sets the default.

-FILE
Specifies that only file values are to be allowed for the access mode argument. If this argument is unspecified, the first access_mode value specified sets the default.

FUNCTION DESCRIPTION:

The SET COMMON ACCESS CONTROL LIST command is used to modify the file or directory common access control list (CACL) of a directory by adding a new entry or changing the access mode of an existing entry. A file CACL contains the access control entries to be applied to all files described in the specified directory. A directory CACL contains the access control entries to be applied to all directories described in the specified directory. See "Function Description" for the Set Access Control List (SA) command for further information.

SET TERMINAL CHARACTERISTICS

Command Name: STTY

Change the file characteristics of a terminal.

FORMAT:

STTY device_name [ctl_arg]

ARGUMENT DESCRIPTION:

device_name
The symbolic name of the terminal as defined at system building (one to six characters).

ctl_arg
One or more arguments chosen from the following list. If an argument is unspecified, the corresponding current value for the file remains in effect.

-LL n
A decimal integer specifying the desired line length. This value excludes the length of the control byte.

-DSW xxxx
A 4-character hexadecimal value specifying the device-specific word. See the *Communications Processing* manual for information on device-specific words.

-DETAB { ON / OFF }
Sets detabbing on or off.

-IN { A / S / N }
Prepares the device to receive asynchronous (A), synchronous (S), or nonbuffered synchronous (N) input.

-OUT { A / S / N }
Prepares the device to transmit asynchronous (A), synchronous (S), or nonbuffered synchronous (N) output.

-TYPE { I / O / B }
Sets the device type to input-only (I), output-only (O), or bidirectional (B).

FUNCTION DESCRIPTION:

The STTY command allows the user to modify the file characteristics associated with a terminal that is not currently reserved. The original file characteristics, established at system building, can be altered to reflect the user's needs. Refer to the *Communications Processing* Manual for details on the use of the STTY command.

Example:

```
STTY  TTYYA  -LL  120  -DSW  910  -IN  S  -OUT  S  -TYPE  B
```

This command sets the file characteristics of the terminal whose device name is TTYYA to a line length of 120 characters, a device-specific word of 910, and a device type of bidirectional. The terminal is to accept synchronous input and send synchronous output.

SORT FILE

Command Name: SORT

Sort the records in a file.

FORMAT:

SORT [ctl_arg]

ARGUMENT DESCRIPTION:

[ctl_arg]
One or more control arguments chosen from the following list:

-IN path
Specifies the name of the file containing the sort descriptors for this sort. If not specified, the user-in file is used.

{-SIZE n
-SZ n}
Indicates the maximum number of 1024-word memory modules to be available to the sort. The value of n can be from 8 to 48 (decimal) for SAF mode and from 8 to 68 (decimal) for LAF mode. An invalid value can cause an illegal memory error code to be displayed. If this argument is not specified, 8 memory modules are used.

-PD
Indicates that a listing of the sort descriptors is to be produced on the user-out file. (Only the first 71 characters of the line will be displayed.) If this argument is not specified, no list is issued.

-FF
When duplicate records are encountered, order them on a first-in/first-out (FIFO) basis.

-DL
When duplicate records are encountered, delete all but one. If -FF has also been specified, all but the first occurrence of the duplicate records will be deleted.

-AK
Output from the sort will be records constructed of the sort keys concatenated in the sequence of specification, prefixed by a record address. The record address is either a simple key or a relative record number, as applicable to the file sorted.

-AD
Output from the sort will be a record constructed of only the record address of its position in the original input file.

FUNCTION DESCRIPTION:

The SORT command provides the capability of sorting a data file according to specifications supplied in a sort descriptor file.

The arguments -AK and -AD cannot both be specified in the same SORT command.

A complete description of the operation and use of the sort component is contained in the *Sort/Merge* manual.

SPAWN GROUP

Command Name: SG

Create, request the execution of, and then delete a task group.

FORMAT:

SG id base_lvl [in_path] [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of the task group to be spawned. It is a 2-character name that cannot have the $ as its first character.

base_lvl
A base priority level, relative to the system level, at which all tasks in this task group will execute. A base level of 0, if specified, is the next higher level above the last system priority level. The sum of the highest system physical level plus 1, and the base level of the group, and the relative level of a task within that group must not exceed 62_{10}.

in_path
The name of the file from which commands and user input are to be read by the task group during its execution. The file name is set to null if the in_path argument is not specified; in_path must be specified if the control argument -ECL (see below) is used or implied.

[ctl_arg]
One or more control arguments chosen from the following list:

-OUT out_path

Defines the pathname of the file which is to receive user output from the task group. If not specified, one of the following assumptions is made:

- If in_path specifies a disk file, out_path is in_path.AO
- If in_path specifies an interactive terminal, out_path is in_path
- If in_path is not specified, out_path is null
- If in_path specifies an input-only device, out_path is null.

-WD path
Specifies that path is to be used as the working directory pathname. This argument is set to null if not specified.

{-EFN root
-EFN root?entry}

The name of a bound unit root entry which is to be the lead task. The root segment name can be suffixed with ?entry, where entry is a symbolic start address within the root segment. If ?entry is not given, the start address established when the bound unit was linked is assumed.

-ECL
The root segment of the command processor is to be loaded as the lead task.

-LRN n
Specifies the highest logical resource number (LRN) which will be referred to by any task in the task group. The maximum value is 252. The default value is the highest LRN used by the system.

-LFN n
Specifies the highest logical file number used by any task in the spawned task group. The maximum value is 255. If -LFN is not specified, n assumes the value 15.

-POOL id

id is a two-character ASCII identifier and is the name of the memory pool from which all memory required by the spawned task group is to be taken. If specified, id must have been defined at system building time. If not, the issuing task group's memory pool is used.

-ARG arg arg . . . arg

Indicates that additional arguments required by the spawned task group during execution follow. These additional arguments are passed to the lead task of the spawned group to be used as necessary, and are substituted for parameters in the command-in file. If used, the -ARG control argument must appear last. Refer to the Appendix A for an explanation of the use of additional arguments.

Note:

-EFN *or* ECL, but not both, can be specified. If neither is specified, -ECL is assumed and the in_path argument is required.

FUNCTION DESCRIPTION:

The SPAWN GROUP command combines the functionality of the CREATE GROUP, ENTER GROUP REQUEST, and DELETE GROUP commands. It implicitly causes the execution of these three functions in sequence (i.e., it allocates and creates the data structures required to define and control the execution of the task group, places a request against the group, thereby activating it, and, when execution terminates, removes all controlling data structures and returns memory used by the task group to the appropriate memory pool).

Because of the sequencing of the functions described above the SG command relieves the user of the issuing task group of the need to be aware of when the spawned task group terminates. The user need take no explicit action to return the terminating group's resources to the system to make them available for use by other task groups. A user may, for example, spawn a task group for another user who wishes to use the Editor or perform a file dump. This task group exists only for the length of time required to perform its function; when it terminates it is deleted automatically.

The issuing task group can itself be a spawned task group, spawned by an operator command or by a command issued by another online task group. In either case, it has the command processor as its lead task.

The SG command can be issued only by an online task group.

SPAWN TASK

Command Name: ST

Create, request the execution of, and then delete a task within the issuing task group.

FORMAT:

ST rel_lvl ctl_arg

ARGUMENT DESCRIPTION:

rel_lvl
The priority level, relative to the task group's base priority level, at which the spawned task is to execute.

ctl_arg
One or more control arguments chosen from the following list:

{-EFN root
-EFN root?entry}
The name of a bound unit root segment which is to be executed. The root segment name can be suffixed with ?entry, where entry is a symbolic start address within the root segment. If no suffix is given, the default start address, established when the bound unit was linked, is assumed.

{-SHARE lrn [ssa]
-SHR lrn [ssa]}
The same bound unit is used as for the task identified by lrn in the same task group. (This task must have been previously defined by a CREATE TASK command specifying this lrn.) ssa is the symbolic start address within the root segment of the task lrn. If none is given, the default start address of the root segment lrn established when it was linked, is assumed.

-WAIT
Specifies that the task issuing this command is to await completion of the spawned task before resuming execution.

-ARG arg arg . . . arg
Indicates that additional arguments required by the spawned task during execution follow. These additional arguments are passed to the spawned task in an extension of the task request block and are substituted for parameters in the command input file. If used, the -ARG control argument must appear last. Refer to Appendix A for an explanation of the use of additional arguments.

Note:
In any invocation of the ST command, -EFN *or* -SHARE, but not both, must be specified.

FUNCTION DESCRIPTION:

The SPAWN TASK command combines the functions of the CREATE TASK, ENTER TASK REQUEST, and DELETE TASK commands in that it constructs all requisite structures for theexecution of the task, activates the task, and then deletes it.

When the spawned task issues a Terminate macro call (see the *System Service Macro Calls* manual), all controlling data structures associated with the task are removed, and the memory they occupied is returned to the task group's memory pool.

A spawned task is not assigned a logical resource number. It is therefore "local" to (i.e., visible only to) the spawning task. It cannot be requested or referred to by any other task, nor can its memory space or code be shared. It can, however, share that of another task which was

assigned an LRN by means of a previously issued CREATE TASK command. The -SHARE control argument indicates that this sharing is to occur.

Multiple tasks can be made to execute concurrently within a given task group by issuing multiple ST commands. Tasks can also be made to execute serially; i.e., one task going to completion before a subsequent task begins execution. The -WAIT control argument is the mechanism which controls concurrency of execution. Judicious use of this argument can also result in a mixture of concurrent and serial execution (see example 3, below)

Example 1:

Three tasks which have no dependencies among them are to be executed. They can be activated concurrently by issuing the following commands:

```
ST  2  -EFN   PROGA
ST  3  -EFN   PROGB
ST  4  -SHARE  10
```

Each of the first two spawned tasks executes its own bound unit in its own memory space. The third shares the code and memory space of a previously created task identified by logical resource number 10. If the task group's base level was specified as 2 when the group was created, the three tasks execute at relative priority levels 4, 5, and 6, respectively.

Example 2:

The three tasks above have dependencies among them which require them to be executed serially. They are activated by the following commands:

```
ST  2  -EFN   PROGA  -WAIT
ST  3  -EFN   PROGB  -WAIT
ST  4  -SHARE  10
```

Tasks 2, 3, and 4 execute sequentially in this example. Since the third task does not specify -WAIT, another activity can be initiated to run concurrently with it.

Example 3:

The first two of the three tasks are unrelated, but there is a dependency between the second and third tasks. The following commands can be used:

```
ST  2  -EFN   PROGA
ST  3  -EFN   PROGB  -WAIT
ST  4  -SHARE  10
```

This sequence causes the first two tasks to be activated to run concurrently. Since the second task specifies the -WAIT argument, it must terminate execution before the third task can begin. The first task may or may not still be running at this time. As in the previous example, another activity can be initiated to execute concurrently with the third task.

STATUS GROUP

Command Name: STG

Display the status of the issuing task group.

FORMAT:

STG [ctl_arg]

ARGUMENT DESCRIPTION:

[ctl_arg]

One or more control arguments chosen from the following list.

-TASKS
Specifies that the status of each task in the issuing task group is to be listed. This is the default if no control arguments are present.

-FILES
Requests the names of all files that are currently associated with the issuing task group, their types, concurrencies, LFNs, and whether they are open or closed.

FUNCTION DESCRIPTION:

The STATUS GROUP command writes to the user-out file a summary of the current status of the issuing task group. In addition to information pertinent to the group as a whole, two other categories of status information are displayed: one relating to tasks within the group and the other relating to files currently associated with the group.

The following items provide status information relative to the task group as a whole:

- Task group identification
- Current state of the task group:
  B — Batch, not rolled out
  A — Active
- Memory pool identification (if not a batch group)
- Current user identification (user_id for batch task group is person.account.ABS)
- Full pathname of error-out file
- Full pathname of user-out file

Task-specific status information consists of the following group of items for each task:

- Task logical resource number (if a created task) or the letters ST (if a spawned task)
- Task priority level
- Current state of the task:
  D — Dormant
  W — Waiting
  A — Active
  X — Being terminated
- First six characters of the task's bound unit name
- Full pathname of the command-in file
- Full pathname of user-in file

File-specific status information consists of the following group of items for each file:

- Full pathname of the file
- Concurrency of the file, represented by a decimal digit in the range 1 through 5. The significance of the digits for the issuing task group and for other task groups is as follows:

Digit	Significance for Issuing Task Group	Significance for Other Task Groups
1	Read only	Read only
2	Read only	Read or write
3	Read or write	No read, no write
4	Read or write	Read only
5	Read or write	Read or write

- File type. The rightmost six bits of the status word form a hexadecimal value for the file type (i.e., the left hexadecimal digit of the formed value can only represent 0 through 3). See the Command In macro call in the *System Service Macro Calls* manual for the file type descriptions.
- Logical file number if one is associated with the file, otherwise spaces.
- Open/closed status of the file; O for open, C for closed.

If there are no files currently associated with the task group, a single item, NO FILES, is returned. The group status information is always returned when this command is used. The task-specific information is returned if no control arguments are given, or if explicitly requested by the -TASKS argument. If the -FILES argument is specified, the file-specific, but not the task-specific, information is given.

TIME

Command Name: TIME

Display the current date and time in ASCII format.

FORMAT:

TIME

ARGUMENT DESCRIPTION:

No arguments are required or permitted with this command.

FUNCTION DESCRIPTION:

The TIME command returns the current date and time of day in an ASCII character string of the form

yyyy/mm/dd hhmm:ss.mmm

yyyy — Year
mm — Month
dd — Day of month
hhmm — Hours and minutes
ss — Seconds
mmm — Milliseconds

The information returned by the TIME command depends on the accuracy of the data entered in the SET DATE operator command.

TAPE POSITIONING

Command Name: TPOS

Position magnetic tape forward or backward to a specific block or file.

FORMAT:

TPOS path [nth] [ctl_arg]

ARGUMENT DESCRIPTION:

path
The path name of the device on which the magnetic tape is mounted.

[nth]
Position the tape to the nth file on the tape, where n is a decimal number. The nth value is optional since it does not apply to all of the accompanying control arguments.

Note:
The default for tape positioning with the TPOS command is file by file. Some of the control arguments in the list below supply alternative positioning capabilities that must be specified.

[ctl_arg]
One or more arguments from those listed below may be chosen. The additional argument must be a qualifier of the initial argument.

-FW n
Space tape forward n files or blocks, where n is a decimal number.

-BK n
Space tape backward n files or blocks.

-BLOCK
Space the tape by block rather than by the default of files.

-TM
Space the tape by tape marks rather than by the default of files.

-RWD
Rewind the tape to BOT (beginning of tape).

-UNL
Unload the tape.

-EOT
Position the tape to the logical EOT (end of tape).

-FF
Position the tape forward, from the *current* position, to the specified file (name).

FUNCTION DESCRIPTION:

The TAPE_POS command offers the capability of positioning a magnetic tape in a variety of ways. In addition to the standard file-by-file positioning procedure, the tape may be positioned to a specific block, tape mark or file name. The facility for positioning tapes by tape mark (-TM) makes it possible to control the positioning of a magnetic tape that has been constructed by other than the standard data management interface. Thus, a user who knows how a tape is constructed in terms of tape mark references can position the tape to any one of these points. Positioning the tape by blocks can be done successfully only within a file. Block processing will not cross tape marks, either forward or backward.

TRANSMIT FILE (LEVEL 6, 66)

Command Name: TRAN

Transmit or receive a file to/from a remote Honeywell Level 6 or 66 system.

FORMAT:

TRAN [ctl_arg]

ARGUMENT DESCRIPTION:

Control arguments are chosen from the following:

mode
A character string composed of up to three characters chosen from the three character sets described below.

I
Specifies that the Level 6 is to act as the initiator.

{S / R}
Specifies whether the Level 6 is to send data or receive data, as follows:
S — Send data
R — Receive data
The default value is R.

{C / D}
Specifies whether the communications line is to remain connected or is to be disconnected upon normal termination of the file transfer, as follows:
C - Line remains connected
D - Line is disconnected
The default value is D.

-Lsys_id
Specifies the remote system involved in the file transfer, as follows:
-L66 — Level 66
-L6 — Level 6

-N {lrn / >SPD>dev_name}
Specifies the logical resource number or pathname used to access the communications line, as follows:
lrn
Logical resource number (LRN) used to access the line.
>SPD>dev_name
Pathname associated with the LRN used to access the line.
The default value is -N 00.

-I {S / R / F} {A / B / 8} Δ pathname
Initiator (Level 6) pathname of file being sent/received (see the *Systems Concepts* manual).

Optional keyword values of the -I argument are:

S — The Level 6 file type is sequential.

R — The Level 6 file type is relative.

F — File type is fixed-relative with nondeletable records.

A — Data type is the 104 ASCII subset.

B — Data type is the 64 ASCII subset.

8 — Data type is binary.

For Level 6 to Level 6 file transmission, the acceptor argument description is as follows:

$$\text{-A}\left\{\begin{matrix}S\\R\\F\end{matrix}\right\}\left\{\begin{matrix}A\\B\\8\end{matrix}\right\} \quad \Delta \quad \text{pathname}$$

Optional keyword values of the -A argument for Level 6 to Level 6 file transmission are identical to those for the -I argument as described above:

For Level 6 to Level 66 file transmission, the argument description is as follows:

$$\text{-A}\left\{\begin{matrix}S\\R\\G\end{matrix}\right\}\left\{\begin{matrix}A\\B\\8\end{matrix}\right\}\left\{\begin{matrix}N\\O\end{matrix}\right\}$$

Optional keyword values of the -A argument for Level 6 to Level 66 file transmission are as follows:

S — The file type on the Level 66 is UFF sequential.

R — The file type is UFF relative (with non-deletable records).

G — The file type is GFRC SSF.

A — Data type is the 104 ASCII subset.

B — Data type is BCD.

8 — Data type is COMP1 or COMP2 binary (8 of 9 bit)

N — The file is to be created (new file) by the acceptor.

O — The file already exists (old file) and is to be sent or received, as appropriate, by the acceptor.

Note:

If neither N nor O is specified, the default value depends on the direction of transfer.

-SR nnnnn

Specifies the record number within the file at which file transmission is to begin. Applies only when a file transmission is restarted. nnnnn is a decimal number from 0 through 99999. The default value is 0.

FUNCTION DESCRIPTION:

See the *Level 6/Level 66 File Transmission* manual for full details on performing a file transfer.

TRANSMIT FILE (BSC 2780/3780)

Command Name: TRANB

Transmit or receive a file to/from a remote nonHoneywell system capable of using the IBM 2780/3780 protocol.

FORMAT:

TRANB [ctl_arg]

ARGUMENT DESCRIPTION:

Control arguments are chosen from the following:

mode
A character string composed of up to three characters chosen from the three character sets described below.

I
Specifies that the Level 6 is to act as the initiator.

{S / R}
Specifies whether the Level 6 is to send data or receive data, as follows:
S — Send data.
R — Receive data.
The default value is R.

{C / D}
Specifies whether the communications line is to remain connected or is to be disconnected upon normal termination of the file transfer, as follows:
C — Line remains connected.
D — Line is disconnected.
The default value is D.

-Lsys_id [path]
Specifies the remote system involved in the file transfer and whether a preliminary transfer file is to initiate the transmission, as follows:
-L278 — NonHoneywell system using IBM 2780 protocol.
-L378 — NonHoneywell system using IBM 3780 protocol.
path — Pathname of the preliminary transfer file to be used to initiate the transfer.

-N {lrn / >SPD>dev_name}
Specifies the logical resource number or pathname used to access the communications line, as follows:
lrn
Logical resource number (LRN) used to access the line.
>SPD>dev_name
Pathname associated with the LRN used to access the line.
The default value is -N 00.

-I {C / P} Δ pathname
Specifies the pathname to be used by the initiator (Level 6) to access the file to be sent or received.

C

Invokes the "cut" facility; splits the file records into 80-character chunks for transmission to a nonHoneywell system.

P

Invokes the "paste" facility; combines the 80-character record chunks received from a nonHoneywell system to form fixed-length records.

FUNCTION DESCRIPTION:

See the *Level 6/BSC2780/3780 File Transmission* manual for full details on performing a file transfer.

TRANSMIT FILE (62, 64; SERIES 200/2000)

Command Name: TRANH

Transmit or receive a file to/from a remote Honeywell Level 6, Level 62, Level 64, or Series 200/2000 system.

FORMAT:

TRANH [ctl_arg]

ARGUMENT DESCRIPTION:

Control arguments are chosen from the following:

mode
A character string composed of up to three characters chosen from the three character sets described below.

{I / A}
Specifies whether this Level 6 is to act as the initiator or the acceptor of the file to be transmitted, as follows:
I — Act as initiator.
A — Act as acceptor.
The default value is A.

{S / R}
Specifies whether this Level 6 is to send data or receive data, as follows:
S — Send data.
R — Receive data.
The default value is R.

{C / D}
Specifies whether the communications line is to remain connected or is to be disconnected upon normal termination of the file transfer, as follows:
C — Line remains connected.
D — Line is disconnected.
The default value is D.

-Lsys_id
Specifies the remote system involved in the file transfer, as follows:
-L62 — Level 62
-L64 — Level 64
-L2K — Series 2000

-N {lrn / >SPD>dev_name}
Specifies the logical resource number or pathname used to access the communications line, as follows:

lrn
Logical resource number (LRN) used to access the line.

>SPD>dev_name
Pathname associated with the LRN used to access the line.

The default value is -N 00.

-I path
Specifies the pathname to be used by the initiator (Level 6) to access the file to be sent or received.

-A path
Specifies the pathname to be used by the acceptor to access the file to be sent or received.

-SR nnnnn
Specifies the record number within the file at which file transmission is to begin. Applies only when a Level 6 to Level 6 file transmission is restarted. nnnnn is a decimal number from 0 through 99999. The default value is 0.

FUNCTION DESCRIPTION:

See the appropriate *File Transmission* manual for full details on performing a file transfer.

WALK SUBTREE

Command Name: WS

Execute command line in specified directory and in all subordinate directories. Print; the path of every directory referenced, on error_out.

FORMAT:

WS path command line [ctl arg]

ARGUMENT DESCRIPTION:

path
The starting node. This must always be the first argument. If -WD is specified, the working directory becomes the starting node.

command line
The command line to be executed. Since the entire command line is treated as a single argument, it must be enclosed in quotes if embedded separators are included.

[ctl_arg]
Control arguments are optional and can appear in any order following the command line.

-FIRST *n*
Makes *n* (where "n" is a decimal number) the first level in the file system hierarchy at which the command line is to be executed. By definition, the normal starting node is level 1.
Default: -FIRST 1

-LAST *n*
Makes *n* the lowest level in the file system hierarchy at which the command line is to be executed.
Default: -LAST 99999, i.e., all levels.

{-BRIEF / -BF}
Suppresses printing to error_out of the names of the directories in which the command line is executed.
Default: Do not suppress printing directory names to error_out

{-BOTTOM-UP / -BU}
Causes execution of the command line to begin at the last level of the storage system hierarchy and to proceed upward to the first level.
Default: Begin at the highest level and proceed downward to the lowest level.

Note:
The WS command establishes a program interrupt handler. If the user "BREAKS" out of the WS command and immediately types PI (program interrupt, see BREAK Procedures in Section 1), his working directory will be changed to the directory he was in when the WS command was typed.

Appendix A

ADDITIONAL COMMAND CONSIDERATIONS

This appendix provides added information in the following areas:

- Additional command line arguments
- Terminal characteristics at login
- Pathname colon convention

ADDITIONAL COMMAND LINE ARGUMENTS (ARG)

A mechanism exists to handle additional arguments entered in command lines dealing with task activation; if the activated task is a user application, the arguments are passed to the task for processing. Parameter substitution of command line arguments is handled for noninteractive command-in files and for user-in files that are the same as noninteractive command-in files.

ARGUMENT PASSING

The arguments following the keyword -ARG in the EGR, EBR, SG, ETR, L, and ST command are passed to the activated task.

When the activated task is the command processor, the argument list is used for parameter substitution.

When the activated task is a program preparation or utility function, the task uses the values in the argument list for its own required arguments.

When the activated task is a user application (that is to be passed arguments), the task must contain an assembly language routine that examines the argument list in the task's parameter block. The parameter block is a variable size augment to the task request block. Parameter blocks and task request blocks are described in the *System Service Macro Calls* manual.

Example 1:

```
EGR AX -WD ^VOL>JR -OUT >SPD>LPT00 -ARG -IN >SPD>CRD00 -LL 80
```

A previous CG command has identified ED (for Editor) as the name of the bound unit root segment to be loaded as the lead task. The arguments supplied to the Editor (viz., input file pathname and maximum line length) are included in the argument list for the EGR command.

Example 2:

```
EGR AX-WD^ VOL>JR-OUT>SPD>LPT00-ARG FILEA >UDD>BOOKS>FILEA-PR 20
```

A previous CG command has identified CPA (the Compare utility program) as the name of the bound unit root segment to be loaded as the lead task. The first and second arguments in the argument list are used for the first and second positional arguments in the Compare command line. The third argument in the list is an optional keyword argument passed to the Compare utility program.

INPUT COMMAND LINE PARAMETER SUBSTITUTION

A substitutable parameter in the command-in file is an ASCII character string in the form &n, where n is one or more digits (e.g., &0, &1, etc.) The digit indicates the position in the argument list of the data element to be substituted. The first argument is substituted for &0,

the second for &1, and so forth. Depending on the case, the first argument can be path or the first additional argument.

If the argument list is smaller than the number of substitutable parameters present in the command-in file, the null parameter is substituted for all parameters not supplied in the argument list. For example, if XY&1Z is the substitutable line, it becomes XYZ after substitution with the null parameter for &1.

Parameter substitution enables a user to change parameters in a noninteractive EC file. For example, this technique can be applied when an EC command line is an abbreviation for a set of parameterized functions. Parameter substitution occurs for all lines read from the command-in file. Parameter substitution also occurs for all lines read from the user-in file when &A is present in the EC file.

Nesting of argument lists is supported when a command line with additional arguments specifies a command file that, in turn, contains a command line with additional arguments that specifies a command file, and so forth. At each level of nesting, the argument list to be used for the parameterized command file is taken from the arguments in the command line that specifies the command file.

EC FILE EXECUTION COMMAND

The EC command has the following format:

EC path [arg_1 arg_2 . . . arg_n]

In the parameterized command-in file, path is substituted for all occurrences of &0, arg_1 is substituted for all occurrences of &1, and so forth. To parameterize directives to system software (e.g., to the Linker) in the EC file, an &A directive must precede the command line that activates the system software. This technique changes the user-in file so that it is identical to the command-in file which is the EC file.

Example 1:

Task group AX has been created previously with the command processor as its lead task. To execute the EC file, the user enters the following command lines:

```
EGR  AX  >SPD>CONSOLE  -WD  ^VOLA>JR
EC  ASM_LNK  TEST  >SPD>LPT01  -NL  TEST.L  START_AD  DIR>SEC
```

The contents of the EC file ASM_LNK.EC are:

```
ASSEM  &1  -COUT  &2  &3
&A
LINKER  &1  -COUT  &4
parameterized linker directives
```

After substitution, the command lines contain:

```
ASSEM  TEST  -COUT  >SPD>LPT01  -NL
&A
LINKER  TEST  -COUT  TEST.L
parameterized linker directives
```

After the EGR command is executed, the user-in and command-in files are the operator terminal. After the EC command is executed, the user-in file is still the terminal, but the command-in file is ASM_LNK.EC. The &A directive in the EC file is required to change the user-in file so that it is the same as the command-in file (ASM_LNK.EC). If the &A directive is not included, Linker directives are read from >SPD>CONSOLE. TEST is substituted for all occurrences of &1, >SPD>LPT01 for &2, -NL for &3, and TEST.L for &4. The first parameter (&0) is not used. The START_AD and DIR>SEC arguments are substituted for parameters in Linker directives.

Example 2:

The command line is:

```
EC  ASMBL  TEST  -COUT  TEST.L  -SAF
```

The contents of the EC file ASMBL.EC are:

```
&PΔSTART  ASSEMBLY  OF  &1
ASSEM  &1  &2  &3  &4  &5  &6  &7  &8
&PΔEND  ASSEMBLY
&QΔ
```

After substitution, the command lines contain:

```
&PΔSTART  ASSEMBLY  OF  TEST
ASSEM  TEST  -COUT  TEST.L  -SAF
PΔEND  ASSEMBLY
&QΔ
```

ASMBL is not substituted for any parameter. Since the first parameter (&1) is a positional parameter, the first argument must always refer to path TEST in the example. Entries for the keyword parameters can appear in any order in the argument list, or not appear at all.

Example 3:

If the command line in Example 2 was mistakenly entered as:

```
EC  ASMBL  TEST  -COUT  -SAF
```

After substitution, the ASSEM command line would contain:

```
ASSEM  TEST  -COUT  -SAF
```

The positional argument following -COUT is missing; the next argument (-SAF) will be substituted in its place, resulting in an incorrect command line.

GROUP ACTIVATION REQUEST COMMANDS

The following commands have parameter substitution performed on the command-in files read by the lead task:

EBR in_path [ctl_arg -ARG arg_1 arg_2. . . arg_n]
EGR id [in_path] [ctl_arg -ARG arg_1 arg_2 . . . arg_n]
SG id base_lvl in_path [ctl_arg -ARG arg_1 arg_2 . . . arg_n]
L [login_id] [destination_id] [ctl_arg -ARG arg_1 arg_2 . . . arg_n]

The lead task can be the command processor or an applications task; each has its own rules for parameter substitution.

If the command processor is the lead task, in_path is substituted for all occurrences of &0, the first argument following -ARG for &1, the second argument for &2, and so forth.

If an applications task is the lead task, the first argument following -ARG is substituted for all occurrences of &0, the second for &1, and so forth.

Example 1:

The command line is:

```
EBR TRYEDT -OUT >SPD>LPT01 -WD ^VOLA>JR -ARG TRYCOM 1 100
```

The contents of the noninteractive file TRYEDT are:

```
&PΔEDIT  COMMANDS  ARE  IN  &0
ED
R  &1
&2,&3
QT
BYE
```

After substitution, the command lines contain:

```
&PΔEDIT  COMMANDS  ARE  IN  TRYEDT
ED
R  TRYCOM
```

1,100P
QT
BYE

The command processor is the lead task. TRYEDT is substituted for &0, TRYCOM for &1, 1 for &2, and 100 for &3.

Example 2:

This example illustrates the nesting of arguments in successive command lines. The task group activation command line is:

```
EGR AX MPG_DATA -WD ^VOLA>JR -ARG TEST -NL TEST.L
```

The first line of the noninteractive command-in file, MPG_DATA, is:

```
EC ASM_LNK &1 >SPD>LPT01 &2 &3 START_AD DIR>SEC
```

The contents of the EC file ASM_LNK.EC are:

```
ASSEM &1 -COUT &2 &3
&A
LINKER &1 -COUT &4
parameterized linker directives
```

After the first substitution, the EC command line is:

```
EC ASM_LNK TEST >SPD>LPT01 -NL TEST.L START_AD DIR>SEC
```

After the next substitution, using the argument list in the EC command, the command lines in ASM_LNK.EC are:

```
ASSEM TEST -COUT >SPD>LPT01 -NL
&A
LINKER TEST -COUT TEST.L
parameterized linker directives
```

In summary, a previous CG command has created the AX task group whose lead task is the command processor. The additional arguments in the EGR command are substituted for the parameters in the command-in file, MPG_DATA. After the EGR command is processed, the command-in file and user-in file is MPG_DATA. After the EC line is processed, the command-in file is ASM_LNK.EC and the user-in file is still MPG_DATA.

Linker reads its directives from the user-in file; if the directives are to be parameterized, they must be in the command-in file. Therefore, the &A directive is used to change the input file read by the Linker from MPG_DATA to ASM_LNK.EC; the arguments in the EC line apply to parameterized lines in the EC file, ASM_LNK.EC.

After the lines in ASM_LNK.EC are processed, the command-in and user-in files revert back to MPG_DATA. For parameterized command lines that follow the first EC line in MPG_DATA, the argument list is that of the EGR command.

TERMINAL CHARACTERISTICS AT LOGIN

The user's terminal can be a noncommunications terminal (MDC-connected) or a communications terminal (MLCP-connected). Both types of terminals are monitored by the listener component of the system. The listener performs certain operations that affect the states of the various types of terminals. Typeouts that appear on the terminal vary according to whether it is a noncommunications or a communications terminal.

NONCOMMUNICATIONS TERMINAL

The following are characteristics of a noncommunications terminal:

- If the terminal is not ready when the listener is activated, no initial output messages are displayed when the terminal comes on line.
- When the listener is activated, direct login terminals that are on line display the message of the day. A task group is spawned for each such terminal, using preset information. If the lead task of the spawned task group is the command processor, the START_UP.EC

file (if present in the user's working directory) is executed. If the lead task is other than the command processor, that task is executed. When the lead task terminates, the message of the day is displayed and a task group is again spawned.

- When the listener is activated, terminals allowing login by abbreviation and terminals requiring the full login command display the message of the day followed by a user login prompter message, identifying the system and giving the data and time, as follows:

  LOGIN system_id -REV rrrr yy/mm/dd hhmm.ss.t

  system_id
  Identification of the system (e.g., GCOS6/MOD 400)

  rrrr
  Software release number (e.g., 0110).

  yy/mm/dd
  Date: yy is last two digits of year; mm is month (from 01 for January to 12 for December); dd is day of month (from 01 to 31).

  hhmm:ss.t
  Time: hh is hour (from 00 to 23); mm is minute (from 00 to 59); ss is second (from 00 to 59); and t is tenths of second (from 0 to 9).

  Once the prompter message is displayed, the user can type in the login command (abbreviation or full command). When the lead task terminates, the message of the day is displayed, followed by the login prompter message.

COMMUNICATIONS TERMINAL

The following are characteristics of a communications terminal:

- Even though a communications terminal is not ready when the listener is activated, a message is displayed when the terminal comes on line.
- When the listener is activated, the same operations are done for communications terminals as are done for noncommunications terminals, with the exceptions noted below for lead task termination.
- When the lead task terminates the session:
  - The LOGOUT message is displayed.
  - A terminal connected by phone and having the hangup option is disconnected. The user must dial in again to use the terminal.
  - A terminal connected through a modem bypass and a terminal connected by phone and not having the hangup option display the message of the day. If the terminal is a direct login terminal, a login task group is spawned. If the terminal allows login by abbreviation or requires a full login, the login prompter message is displayed.

PATHNAME COLON CONVENTION

The pathname colon convention allows the user to employ the GET or CREATE FILE command to append additional ASCII characters to a pathname that has been previously associated with an LFN (logical file number) through the ASSOCIATE PATH command.

The pathname associated through the ASSOCIATE PATH command will be completed when a CREATE FILE or GET command is issued using the colon (:) option in the path argument. The effect of the colon is to cause the system to replace the colon by the previously associated pathname (i.e., to perform character string concatenation).

The following examples illustrate the use and effect of the colon option. The examples assume that the user's working directory is ^SYSO1>USERA when the ASSOCIATE PATH command is issued and is ^SYSO1>USERB when the GET or CREATE FILE command is issued.

Associated Pathname	GET/CREATE Pathname	Result
none	:OLD>DELA	^SYS01>USERB>OLD>DELA
none	OLD>DELA	^SYS01>USERB>OLD>DELA
none	none	^SYS01>USERB
none	:Δ	^SYS01>USERB
DELA	none	^SYS01>USERA>DELA
DEL	:B	^SYS01>USERA>DELB
^VOL1>UDD>FILE1	:Δ	^VOL1>UDD>FILE1
^VOL2>	:FILE02	^VOL2>FILE02
^VOL2	:>FILE02	^VOL2>FILE02

Appendix B

Intersystem Link (ISL) Directives

An Intersystem Link (ISL) is a hardware element interconnecting two busses, thereby permitting the same functions between two units on different busses as between two units on the same bus. For example, an ISL can provide shared memory capability, central processor to central processor interrupts, dual access to controllers, or simple bus extention. Linked systems can contain multiple busses linked by multiple ISLs. This appendix describes the ISL directives. The ISL system building procedures are described in the *System Building* manual. For a description of the ISL hardware, see the *Minicomputer Handbook*.

ISL directives enable a user to specify the settings for the ISL masks and tables for each ISL twin. The ISL Configurator, which is a software component initiated through a command, processes these directives and writes to a file (ISL_ROUTINEx) the ISL loader program that contains the I/O orders necessary to load the ISL masks and tables. The ISL Configurator does not load the ISL; it generates the ISL loader.

ISL LOADER FILE CREATION

The ISL loader is in a file that is created by the ISL Configurator (assuming that the file does not already exist). If the ISL Configurator creates the file, the default file size is 15 records, each of 256 bytes.

A user has the option to create the ISL file using the CREATE FILE utility described in Section 2 of this manual. The create file command is on one of the forms

```
CF ^vol_id>ISL_ROUTINES -F_REL [ctl_arg]
```

for SAF mode or

```
CF-^ vol_id>ISL_ROUTINEL -F_REL [ctl_arg]
```

for LAF mode.

The file ISL_ROUTINEx is in the major directory of the specified volume. It must be a fixed relative file with a minimum allocation of five records specified through the control arguments. If any ISLs are to be dumped, three additional records are required. Additional file space can be estimated from the number of words needed to implement each of the following ISL configuration directives:

ISL	— 6 words
LMEM+/RMEM+	— 6 words
LMEM−/RMEM−	—12 words
LCHAN/RCHAN	— 6 words
LCP+/RCP+	—12 words
LCP−/BCP−	—12 words

ISL CONFIGURATOR

The ISL Configurator is a software component that runs under the operating system. It reads ISL directives from the user-in file and generates an ISL loader. The Configurator is invoked by the ISLCON command described below.

ISLCON

Command Name: ISLCON

Generate an ISL loader to load Intersystem Link (ISL) address maps and masks.

FORMAT:

```
ISLCON path [{-SAF}]
            [{-LAF}]
```

ARGUMENT DESCRIPTION:

path

The pathname of the volume on which the ISL loader file is to be written.

FUNCTION DESCRIPTION:

The ISLCON command is used to invoke the ISL Configurator which obtains ISL configuration directives from the user-in file. It generates an executable loader program that will load ISL address maps and masks during ISL configuration. The ISL loader is created in the volume major directory specified in the path argument.

Example:

ISLCON ^V20022>ISL_ROUTINES

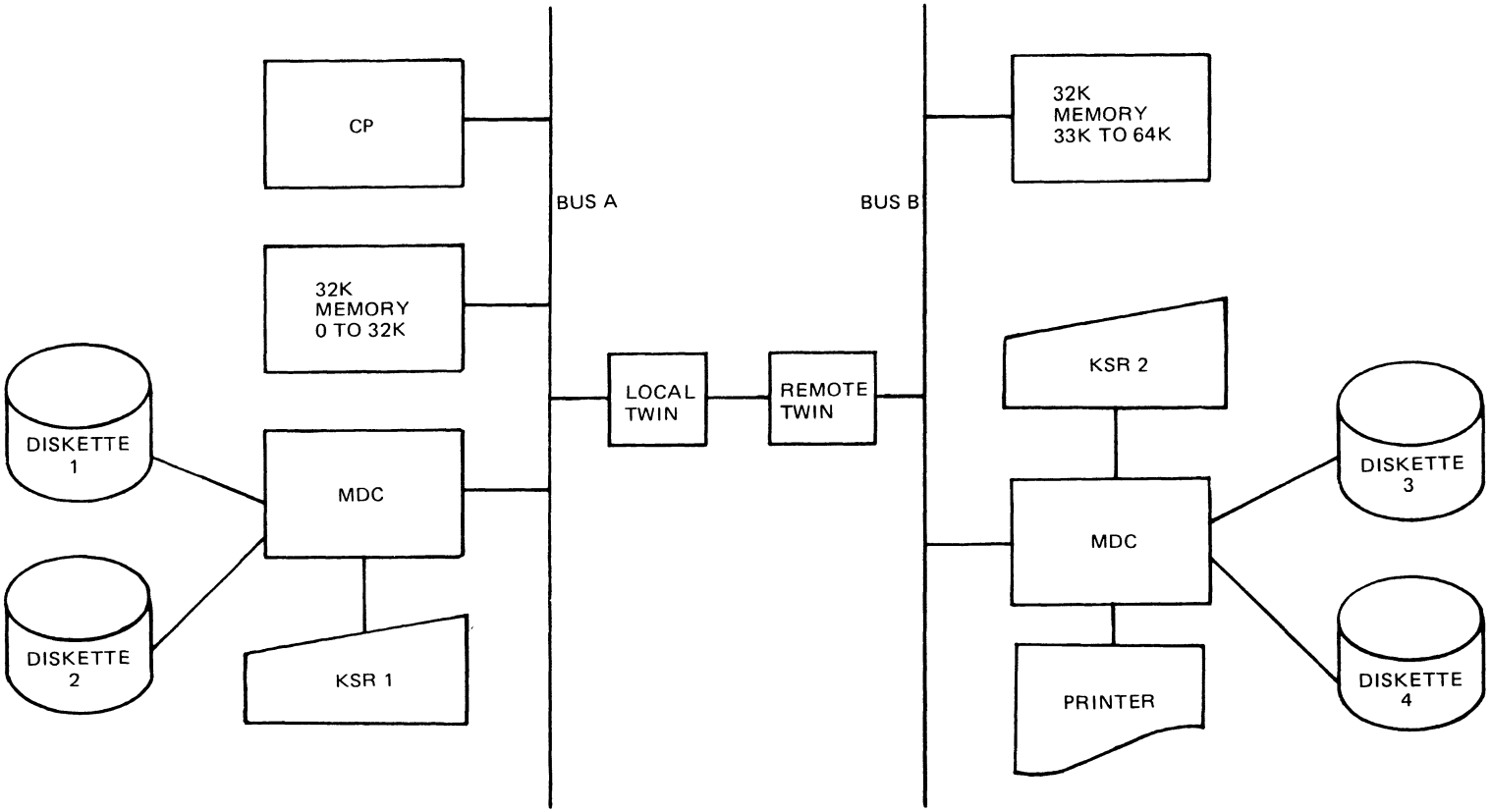
Generation of the ISL loader is to be done. ISL configuration directives are obtained from the user-in file and written to the file ^V20022>ISL_ROUTINES.

SAMPLE INTERSYSTEM LINK

Figure B-1 shows a sample Intersystem Link defined by the ISL configuration directives. Figure B-2 is a diagram of a possible hardware configuration corresponding to the ISL directives.

```
ISLCON ^ISLIN-SAF
?
ISL X'2000',DUMP
?
DUMP X'1380'
?
LMEM+ 4,0
?
LMEM+ 5,1
?
LMEM+ 7,3
?
LMEM+ 6,2
?
LMEM- 0,4
?
LMEM- 1,5
?
LMEM- 2,6
?
LMEM- 3,7
?
LCHAN X'0600'
?
LCHAN X'0680'
?
LCHAN X'0700'
?
LCHAN X'0780'
?
RCP+ X'0040',X'0000'
?
QUIT
```

Figure B-1. Sample Intersystem Link



DEVICE	CHANNEL NO.
DISKETTE 1	0400
DISKETTE 2	0480
DISKETTE 3	0600
DISKETTE 4	0680
KSR 1	0500
KSR 2	0700
PRINTER	0780
ISL	2000, 2001

Figure B-2. ISL Hardware Configuration

ISL CONFIGURATION DIRECTIVES

The ISL Configurator interprets the following ISL configuration directives which are used to initialize ISLs, and designate central processors, memory addresses and channel numbers on local and remote busses linked by ISL twins:

- ISL (Initializes an ISL)
- LMEM (Indicates a desired memory address on a local bus)
- RMEM (Indicates a desired memory address on a remote bus)
- LCHAN (Indicates a desired channel number on a local bus)
- RCHAN (Indicates a desired channel number on a remote bus)
- LCP (Indicates a desired central processor on a local bus)
- RCP (Indicates a desired central processor on a remote bus)
- DUMP (Indicates channel number to which ISL address maps and masks are to be printed)
- QUIT (Indicates the end of ISL configuration input stream)

ISL

Directive Name: ISL

The ISL (Intersystem Link) directive causes an ISL (consisting of local and remote ISL twins), on a designated channel, to be initialized. There is one ISL directive for each ISL (i.e., pair of twins).

FORMAT:

ISL X'channel'[,DUMP]

ARGUMENT DESCRIPTION:

X'channel'

Specifies a four digit hexadecimal channel number (i.e., 16-bits) for both local and remote ISL twins constituting an ISL. The actual channel number occupies only the high-order 10 bits (i.e., bits 0 through 9).

[DUMP]

Causes the ISL loader to print information loaded into ISL masks and address tables after configuration. A maximum of ten ISLs can be dumped.

FUNCTION DESCRIPTION:

All memory and channel specifications following the ISL directive refer to the designated ISL on its respective channel. There is no limit on the number of ISLs that can be configured.

Initially, channel and memory hit bits and channel translate table elements are set to off (i.e., 0); however, memory address translate table elements are set to on (i.e., 1).

An ISL(s) twin(s) connected to a local bus must be configured prior to a twin(s) connected to a remote bus because a channel on a remote bus cannot be referenced without having the appropriate hit bit set in the channel number RAM (Random Access Memory) of the local ISL twin.

Example:

ISL X'2000', DUMP

DUMP

Directive Name: DUMP

Specifies the channel number to which the contents of ISL masks and address maps will be printed.

FORMAT:

DUMP X'nnnn'

ARGUMENT DESCRIPTION:

X'nnnn'

nnnn is a four digit hexadecimal channel number (i.e., 16 bits) used to indicate the channel number to which the contents of the ISL address maps and masks will be printed. It must represent either a printer or KSR that uses the first character of its buffer as a control character.

FUNCTION DESCRIPTION:

ISL directives that specify the DUMP argument will be dumped to the channel specified in the DUMP directive. If this is to occur, there must be one and only one DUMP directive. Only ten ISLs can be dumped.

Example:

```
ISL X'0600', DUMP
  .
  .
  .
DUMP X'1380'
```

In this example, a local and remote ISL pair are assigned to channel 0600_{16} (with the DUMP option) through the ISL directive. The contents of this ISL pair's address maps and masks will be printed on a line printer assigned to channel 1380_{16}.

LCHAN, RCHAN

Directive Name: LCHAN, RCHAN

Sets to "on" the channel hit bit corresponding to the desired channel on either the local or remote twin.

FORMAT:

$\left\{\begin{matrix}\text{LCHAN}\\ \text{RCHAN}\end{matrix}\right\}$ X'nnnn'

ARGUMENT DESCRIPTION:

LCHAN

Indicates a channel whose hit bit is to be set "on" in the local twin.

RCHAN

Indicates a channel whose hit bit is to be set "on" in the remote twin.

X'nnnn'

nnnn is a four digit hexadecimal channel number (i.e., 16 bits) for either a local or remote ISL twin. The actual channel number is indicated only in the high-order 10 bits (i.e., bits 0 through 9 of the 16-bit hexadecimal channel number nnnn).

FUNCTION DESCRIPTION:

This directive causes the channel hit bit, specified by the high-order 10 bits of the 16-bit channel number, to be set to 1 (i.e., on) in the channel number RAM (Random Access Memory) of either the local ISL twin or the remote ISL twin (depending on choice of either the LCHAN or the RCHAN directives, respectively).

Example:

LCHAN X'0600'

Bits:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

High-order 10 bits

In this example, the channel hit bit will be set to 1 (i.e., on) in entry 24 of the local channel number RAM of the local ISL twin.

LCP, RCP

Directive Name: LCP, RCP

Sets the channel hit bits and channel address translate values for central processors on either local or remote busses.

FORMAT:

$$\left\{\begin{array}{l}\text{LCP+}\\ \text{RCP+}\\ \text{LCP}-\\ \text{RCP}-\end{array}\right\}\ \text{X'nnnn', X'cccc'}$$

ARGUMENT DESCRIPTION:

LCP
Indicates a CP to be entered into the local twin.

RCP
Indicates a CP to be entered into the remote twin.

X'nnnn'
nnnn is a four digit hexadecimal channel number (i.e., 16 bits) for either a "local" or "remote" associated ISL "twin" associated with either a "remote" or "local" CP. The actual channel number is indicated only in the high-order 10 bits (i.e., bits 0 through 9 of the 16-bit channel number nnnn) and is left-justified. nnnn must be in the range of 0000_{16} to $000F_{16}$.

X'cccc'
cccc is a four digit hexadecimal number (i.e., 16 bits) representing a value setting for the channel translate table cell associated with channel nnnn. cccc must be in the range of 0000_{16} to $000F_{16}$.

FUNCTION DESCRIPTION:

The LCP+ directive causes the channel hit bit, specified by the high-order 10 bits of the 16-bit hexadecimal channel number, to be set to 1 (i.e., on); it also causes the channel translate table cell (associated with the channel number for either a local or remote CP) to be set to a specified value. RCP+ performs the same function for the remote twin.

The LCP− directive causes the channel hit bit, specified by the high-order 10 bits of the 16-bit hexadecimal channel number, to be set to 0 (i.e., off); it also causes the channel translate table cell (associated with the channel number for either a local or remote CP) to be set to a specified value. The only instance in which LCP− might be specified without a corresponding RCP+ specification involves a system in which an IOLD from on CP is intended to cause an interrupt to a CP different from the issuing CP upon completion of the data transfer. RCP− performs the same function for the remote twin.

Example:

LCP X'0040', X'0000'

Bits:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

High-order 10 bits

In this example, the binary value of bit 9 (i.e., 1) determines that the channel hit bit will be set to 1 (i.e., on) in entry 1 of the channel number RAM of the local ISL twin associated with the local CP. The channel translate table cell (associated with the channel number for the local CP) will be set to a value of 0000.

LMEM, RMEM

Directive Names: LMEM, RMEM

Indicate a desired memory address on local and remote busses by setting a hit bit, which indicates the desired address, in a memory mask RAM.

FORMAT 1:

LMEM+ hhhh,tttt

FORMAT 2:

LMEM- hhhh,tttt

FORMAT 3:

RMEM+ hhhh,tttt

FORMAT 4:

RMEM- hhhh,tttt

ARGUMENT DESCRIPTION:

LMEM+

Indicates that the specified hit bit in the memory mask RAM of the local twin is to be set on and the memory address translate table element is to be set to the specified value.

LMEM-

Indicates that the specified hit bit in the memory mask RAM of the local twin is to be set off and the memory address translate table element is to be set to the specified value.

RMEM+

Same as LMEM+ except that the action occurs at the remote twin.

RMEM-

Same as LMEM- except that the action occurs at the remote twin.

hhhh

Hit bit in memory mask RAM of local or remote twin; is a decimal value from 0 through 1023.

tttt

Address value of memory address translate table element in associated local or remote twin; is a decimal value from 0 through 1023.

FUNCTION DESCRIPTION:

The LMEM+ and RMEM+ directives set the hit bits and values in the local or remote twin exactly as specified. The LMEM- and RMEM- directives not only set hit bits and values in the twin specified but also can set a "mirror image" of the hit bits and values in the other twin. The hit bit specified in one twin is made the memory address translate value for the other twin; the memory address translate value for this twin is made the hit bit for the other twin. This feature is designed for passing dummy memory address values to the other twin. To prevent the "mirroring effect", before a LMEM- is issued, the memory address table element in the other twin must first be set to a value different from 3FF.

The values for hhhh and tttt are decimal from 0 through 1023.

FORMAT 1:

This form of the directive causes the hit bit represented by hhhh in the memory mask RAM (Random Access Memory) of a local ISL twin to be set to 1 (i.e., on) and the memory address translate table element, associated with memory mask RAM, will be set to the value represented by tttt.

Example:

LMEM+ 1,5

In this example, hit bit 1 is set to 1 (i.e., on) by substituting the value 1 for hhhh and indicating the plus (+) sign, immediately following the LMEM directive, which sets hit bit 1 on. The memory address translate table element is set to an address value of 5 by substituting the value 5 for tttt.

FORMAT 2:

This form of the directive causes the hit bit represented by hhhh in the memory mask RAM (Random Access Memory) of a local ISL twin to be set to 0 (i.e., off) and the memory address translate table element, associated with the memory mask RAM, will be set to the value represented by tttt.

If the remote ISL twin's hit bit (represented by tttt, in the case of the remote ISL twin) is 0 (i.e., off) and its memory address translate table element is an initialized state (i.e., all 1 bits or $3FF_{10}$), the hit bit tttt will be set to 1 (i.e., on) and the memory address translate table element tttt will be set to hhhh in the remote ISL twin.

Example:

LMEM– 7,3

In this example, the local ISL hit bit 7 is set to 0 (i.e., off) by substituting the value 7 for hhhh and indicating the minus (–) sign, immediately following the LMEM directive, which sets hit bit 7 to off. The local ISL memory address translate table element is set to an address value of 3 by substituting the value 3 for tttt. If the remote ISL twin's hit bit (represented by ttt, in the case of the remote ISL twin) is 0 (i.e., off) and its memory address translate table element is in an initialized state (i.e., all 1 bits or $3FF_{16}$) hit bit 3 of the remote ISL twin will be set to 1 (i.e., on) and the memory address table translate element will be set to 7 in the remote ISL twin. This action involving the other ISL twin is referred to as "mirroring".

FORMAT 3:

This form of the directive causes the hit bit represented by hhhh in the memory mask RAM (Random Access Memory) of a remote ISL twin to be set to 1 (i.e., on) and the memory address translate table element, associated with the memory mask RAM, to be set to the value represented by tttt.

Example:

RMEM+ 0,4

In this example, hit bit 0 is set to 1 (i.e., on) by substituting the value 0 for hhhh and indicating the plus (+) sign, immediately following the RMEM directive, which sets hit bit 0 on. The memory address translate table element is set to an address value of 4.

FORMAT 4:

This form of the directive causes the hit bit represented by hhhh in the memory mask RAM (Random Access Memory) of a remote ISL twin to be set to 0 (i.e., off) and the memory address translate table element, associated with the memory mask RAM, to be set to the value represented by tttt.

If the local ISL twin's hit bit (represented by tttt, in the case of the remote ISL twin) is 0 (i.e., off) and its memory address translate table element is in an initialized state (i.e., all 1 bits or $3FF_{16}$), the hit bit tttt will be set to 1 (i.e., on) and the memory address translate table element tttt will be set to hhhh in the local ISL twin.

EXAMPLE:

RMEM– 2,6

In this example, the remote ISL hit bit 2 is set to 0 (i.e., off) by substituting the value 2 for hhhh and indicating the minus (–) sign immediately following the RMEM directive which sets hit bit 2 off. The remote ISL memory address translate table element is set to an address value of 6.

If the local ISL twin's hit bit (represented by tttt) is 0 (i.e., off) and its memory address translate table element is in an initialized state (i.e., all 1 bits or $3FF_{16}$), hit bit 6 of the local ISL twin will be set to 1 (i.e., on) and the memory address table translate element will be set to 2 in the local ISL twin. This action involving the other ISL twin is referred to as "mirroring".

QUIT

Directive Name: QUIT

The QUIT directive is the last configuration directive in the user input file.

FORMAT:

QUIT

ARGUMENT DESCRIPTION:

Not applicable.

FUNCTION DESCRIPTION:

When this directive is encountered, it signifies the end of the ISL configuration input stream; the ISL configurator terminates.

Appendix C

File Change Directives

The File Change (FC) command is used to change the contents of a disk file. Changes can be made to a sector or control interval in the named file.

The FC command invokes the FC processor, which reads directives entered from the terminal. Each FC directive is summarized below and described in detail later in this appendix.

Directive Name	Function
R	Read a specified sector or control interval.
P	Print the contents of the last sector or control interval read.
C	Verify the contents of specified locations in the specified sector or control interval. Place the new values of these locations in an FC processor buffer.
W	Write the new values in the FC processor buffer to the specified locations in the sector or control interval.
Q	Terminate execution of the FC processor.

FILE CHANGE COMMAND

The FC processor is invoked by the FC command.

FORMAT:

FC path

ARGUMENT DESCRIPTION:

path
Name of the file to be changed. A peripheral device pathname indicates sectors are to be changed. A file pathname indicates control intervals are to be changed.

EXAMPLE:

```
FC >SPD>DSK03
```

This command invokes the FC processor to apply changes to sectors on disk DSK03.

```
FC ^VOLO1>UDD>JONES>FILE_A
```

This FC command invokes the FC processor to apply changes to control intervals in file FILE_A.

FILE CHANGE DIRECTIVES

Once the FC processor is invoked, the user can issue the File Change directives from his terminal.

Each FC directive consists of a directive name only or a directive name followed by one or more arguments. If more than argument is to be specified in a single directive, the directive name must be followed by a space and each argument (except the last) must be followed by a comma.

The FC directives can be entered in any order, except for the Quit directive, which must be entered last. Normally, the directives are entered in the order Read, Print, Change, and Write.

READ

The Read directive (R) reads a sector or control interval of the file specified in the path argument of the FC command. The sector or control interval specified in the Read directive is the sector or interval that will be operated upon by the following directives until another Read directive is issued or the Quit directive is entered.

FORMAT:

R adr_1

ARGUMENT DESCRIPTION:

adr_1
Specifies a relative sector or control interval number within the file; is expressed as one to six right-justified hexadecimal digits.

EXAMPLE:

R A

Read the tenth sector or control interval in the file named by the path argument of the FC command.

PRINT

The Print directive (P) causes the printing of the contents of the sector or control interval read by the last read directive. The sector or control interval contents are placed on the user_out file.

FORMAT:

P

ARGUMENT DESCRIPTION:

No arguments are used with this directive.

EXAMPLE:

P

If the last Read directive was R A, print the contents of the tenth sector or control interval of the file named in the path argument of the FC command.

CHANGE

The Change directive (C) is used to verify the current value of the specified location in the sector or control interval and then replace that value with a new value. The sector or control interval whose location(s) is to be changed is that sector or control interval specified in the preceding Read directive.

Each new value is stored in a buffer in the FC processor. The location(s) in the control interval or sector are not overwritten until a Write directive is given (see "Write" below).

Within a single Change directive, verification values can be specified for any or all locations. If any verification value fails to match the value at its location, none of the changes are placed in the FC processor buffer.

FORMAT:

C /adr_1,($verval_1$,$newval_1$[, . . . , $verval_n$,$newval_n$]) [,/adr_2,
($verval_1$,$newval_1$ [, . . . , $verval_n$,$newval_n$]) , . . . ,/adr_n,($verval_1$,$newval_1$
[, . . . , $verval_n$,$newval_n$])]

Note:

One or more lines of arguments may be specified. When two or more lines of arguments are entered, the last character on each line must be a valid hexadecimal character. Individual fields, values, and addresses must not be split between lines.

ARGUMENT DESCRIPTION:

/adr$_n$

Relative location at which the first (or only) change value will be applied. Each address consists of one to four right-justified hexadecimal characters. Each address must be preceded by the / (stroke) character. The relative location must not exceed the size of the sector or control interval. Subsequent change values (if any) are applied to succeeding locations.

verval$_n$

Verification value; one to four hexadecimal characters specifying the value that should now be present in the location to which the change will be made. Each verification value must be immediately followed by a change value (see the newval$_n$ argument below). The verification value(s) and the change value(s) associated with each address value must be enclosed in parentheses.

newval$_n$

The new value (change value) to be inserted at a location, thereby replacing the contents of that location. The value must be from one to four hexadecimal characters.

EXAMPLE:

```
C  /25,(42,52),/31,(4D,4F,37,39,41,42)
```

Assuming that this is the tenth control interval of the file named by the argument of the FC command, change position 25_{16} from B to R, position 31_{16} from M to O, position 32_{16} from 7 to 9, and position 33_{16} from A to B.

WRITE

The Write directive (W) is used to write the current buffer to the sector or control interval position(s) in the file. All changes to sectors or control intervals specified in the preceding Change directives are stored in a buffer in the FC processor. The Write directive causes these changes to be written in the specified locations in the sectors or control intervals. A Write directive must be issued before changes are made to a new sector or control interval.

FORMAT:

```
W
```

ARGUMENT DESCRIPTION:

No arguments are used with this directive.

EXAMPLE:

```
W
```

Assuming that the Change directive was

```
C  /25,(42,52),/31,(4D,4F,37,39,41,42)
```

The W directive causes position 25_{16} to be change to R, 31_{16} to M, 32_{16} to 9, and 33_{16} to B.

QUIT

The Quit directive (Q) informs the FC processor that no further directives will be forthcoming. This directive causes execution of the FC processor to terminate. The user is returned to command level.

FORMAT:

```
Q
```

ARGUMENT DESCRIPTION:

No arguments are used with this directive.

SAMPLE FILE CHANGE COMMANDS

In the following example, the volume label of a cartridge disk is to be changed.

```
FC  >SPD>DSK03          (Sector change)
R  7                    (Read sector 7)
P                       (Print sector 7 on user-out file)
C  /A,(30,31)           (Change tenth location from 0 to 1)
W                       (Write change to sector 7)
Q                       (Terminate)
```

In the following example, changes are to be made to data file FILE_A, which has 100 control intervals, each with 240 positions. The second and tenth control intervals are to be changed. In both intervals, the first position should be O instead of A.

```
FC   VOL01>UDD>JONES>FILE_A     (Control interval change)
R  2                            (Read second control interval)
P                               (Print second control interval
                                 on user-out file)
C  /1,(41,30)                   (Change A to O)
W                               (Write change)
R  A                            (Read tenth control interval)
P                               (Print tenth control interval
                                 on user-out file)
C  /1,(41,30)                   (Change A to O)
W                               (Write change)
Q                               (Terminate)
```

Appendix D

ASCII and EBCDIC Character Sets

Tables D-1 and D-2 illustrate the ASCII and EBCDIC character sets, respectively. In addition to the ASCII characters, Table D-1 shows the hexadecimal equivalents; Table D-2 shows the binary and hexadecimal equivalents of the EBCDIC character set.

Following are lists of the control characters and special graphic characters that appear in the two tables:

Control Characters

ACK	Acknowledge	EOT	End of Transmission	PF	Punch Off
BEL	Bell	ESC	Escape	PN	Punch On
BS	Backspace	ETB	End of Transmission Block	RES	Restore
BYP	Bypass	ETX	End of Text	RLF	Reverse Line Feed
CAN	Cancel	FF	Form Feed	RS	Reader Stop
CC	Cursor Control	FS	Field Separator	SI	Shift In
CR	Carriage Return	GE	Graphic Escape	SM	Set Mode
CU1	Customer Use 1	GS	Group Separator	SMM	Start of Manual Message
CU2	Customer Use 2	HT	Horizontal Tab	SO	Shift Out
CU3	Customer Use 3	IFS	Interchange File Separator	SOH	Start of Heading
DC1	Device Control 1	IGS	Interchange Group Separator	SOS	Start of Significance
DC2	Device Control 2	IL	Idle	SP	Space
DC3	Device Control 3	IRS	Interchange Record Separator	STX	Start of Text
DC4	Device Control 4	IUS	Interchange Unit Separator	SUB	Substitute
DEL	Delete	LC	Lowercase	SYN	Synchronous Idle
DLE	Data Link Escape	LF	Line Feed	TM	Tape Mark
DS	Digit Select	NAK	Negative Acknowledgment	UC	Uppercase
EM	End of Medium	NL	New Line	US	Unit Separator
ENQ	Enquiry	NUL	Null	VT	Vertical Tab
EO	Eight Ones				

Special Graphic Characters

¢	Cent Sign	?	Question Mark
.	Period, Decimal Point	`	Grave Accent
<	Less-than Sign	:	Colon
(	Left Parenthesis	#	Number Sign
+	Plus Sign	@	At Sign
	Logical OR	'	Prime, Apostrophe
&	Ampersand	=	Equal Sign
!	Exclamation Point	"	Quotation Mark
$	Dollar Sign	~	Tilde
*	Asterisk	{	Opening Brace
)	Right Parenthesis		Hook
;	Semicolon		Fork
	Logical NOT	}	Closing Brace
-	Minus Sign	\	Reverse Slant
/	Slash		Chair
			Vertical Line		Long Vertical Mark
,	Comma	[	Opening Bracket
%	Percent	]	Closing Bracket
_	Underscore	^	Circumflex
>	Greater-than Sign		

TABLE D-1. ASCII/HEXADECIMAL EQUIVALENTS

	H1							
H2	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

TABLE D-2. EBCDIC/HEXADECIMAL/BINARY EQUIVALENTS

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	00				01				10				11				Bit Positions 0, 1
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2, 3
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
0000	0	NUL	DLE	DS		SP	&	–						{[a]	}[a]	\[a]	0	
0001	1	SOH	DC1	SOS				/		a	j	~[a]		A	J		1	
0010	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	
0011	3	ETX	TM							c	l	t		C	L	T	3	
0100	4	PF	RES	BYP	PN					d	m	u		D	M	U	4	
0101	5	HT	NL	LF	RS					e	n	v		E	N	V	5	
0110	6	LC	BS	ETB	UC					f	o	w		F	O	W	6	
0111	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7	
1000	8	GE[a]	CAN							h	q	y		H	Q	Y	8	
1001	9	RLF[a]	EM						\[a]	i	r	z		I	R	Z	9	
1010	A	SMM	CC	SM		¢	!	][a]	:								][a]	
1011	B	VT	CU1[a]	CU2[a]	CU3[a]	.	$	,	#									
1100	C	FF	IFS		DC4	<	*	%	@					[a]		[a]		
1101	D	CR	IGS	ENQ	NAK	(	)	–	'									
1110	E	SO	IRS	ACK		+	;	>	=					[a]				
1111	F	SI	IUS	BEL	SUB	'		?	"								EO[a]	

[a]This character is not supported in the 2780 character set.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) GCOS 6 COMMANDS	ORDER NO.	CB02, REV. 1
		DATED	JUNE 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20965, 3678, Printed in U.S.A.

CB02, Rev. 1