Honeywell

TIME SHARING SYSTEM REFERENCE MANUAL

SERIES 60 (LEVEL 66)/6000

SOFTWARE

Honeywell

TIME SHARING SYSTEM REFERENCE MANUAL

SERIES 60 (LEVEL 66)/6000

SOFTWARE

SUBJECT:

General Description of the Time Sharing System (TSS) including the Command Language, Files, Terminal Usage, BASIC, FORTRAN, Text Editor and Service Subsystem.

SPECIAL INSTRUCTIONS:

This manual replaces the following manuals:
Time Sharing BASIC, DD16
TSS System Programmers Reference Manual, DD17
Time Sharing Text Editor, DD18
TSS Terminal/Batch Interface, DD21
TSS General Information, DD22

SOFTWARE SUPPORTED:

SERIES 60 LEVEL 66 SOFTWARE RELEASES 4S2 and DPS1.2
SERIES 6000 SOFTWARE RELEASE JS2

DATE:

October 1979

ORDER NUMBER:

DJ31-00

PREFACE

This manual provides overall information on the Time Sharing System (TSS). Included in this document are command language, time sharing file description, terminal usage, system programming, test and debug aids, BASIC, text editor, FORTRAN, error message definitions, and site administrator tools.

The usage of temporary user files is described on the basis of what is done by the Honeywell-supplied TSS subsystems, primarily BASIC and EDITOR, in Section II.

Section III contains general descriptions of and operational procedures for several remote terminals. Paper tape preparation and reading are included. Methods of correcting operator typing errors are described.

The full complement of supplied commands available to a user is described in alphabetic order in Section IV.

Section VI describes how the user can build subsystem programs to operate in the time sharing environment using derail (DRL) instructions which are equivalent to master-mode entry (MME) functions in the batch mode.

Section VII describes the loading of subsystem programs and testing them using TSS debug trace package.

BASIC (Beginner's All-purpose Symbolic Instruction Code) is a problem-oriented, algebraic programming language that enables the user to present his program in ordinary mathematical notation, with simple and precise vocabulary and grammar.

The text editor subsystem allows a user to build a text file, append to an existing file, and edit a file.

The time sharing system FORTRAN is described in Section XI.

The error messages generated by the time sharing system are documented in Section XII.

Section XIII describes the subsystems and function available to site administrators to allow them to customize the time sharing system.

File No.: 1713 1P13

DJ31-00

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

ILLUSTRATIONS

TABLES

SECTION I

INTRODUCTION

The Time Sharing System (TSS) operates under the direction of the General Comprehensive Operating Supervisor (GCOS), and constitutes one dimension of an integrated, multidimension information processing system. Under GCOS, the multiprocessing dimensions carry on their activities simultaneously, with intercommunication existing between all processing dimensions. This intercommunication feature has considerable significance for the user of a time sharing terminal.

The Time Sharing System (TSS) consists of a Time Sharing Executive, a number of independent processing subsystems which operate under the Executive, and a common command language. The major subsystems of the Time Sharing System include the following:

- ABACUS -- A desk calculator facility featuring complex algebraic capabilities such as functions, summation operations, and remembered variables.
- BASIC -- An algebraic-language compiler/executor designed for the user with numerical calculations involving relatively small quantities of data.
- CONVERT -- A facility for submitting a punch card format job at a time sharing terminal for processing as a batch job. Job status is available on request. The JOUT subsystem complements the batch submission facilities of JRN by providing the capability to scan the job output.
- dataBASIC -- Honeywell's dataBASIC subsystem provides for data base management and inquiry by combining data base manipulation capabilities with a BASIC type language. It permits a file to be constructed, maintained, retrieved, and deleted on a content-addressable basis.
- TEXT EDITOR (and RUNOFF) -- A facility for building, maintaining, and reformatting text files.
- TSS ALGOL -- An ALGOL subsystem that gives the time sharing user access to the capabilities of the ALGOL language.
- TSS JOVIAL -- A JOVIAL subsystem that provides the time sharing user with access to the capabilities of the JOVIAL language processor.
- TSS FORT -- A time sharing based FORTRAN subsystem. Refer to the FORTRAN manual.
- TSS YFORT -- A subsystem interface to the batch-based FORTRAN compiler. Refer to the section on FORTRAN or to the <u>FORTRAN Reference Manual</u>, DG75.

The following subsystems and subroutines provide service and utility functions for the Time Sharing System:

- ACCESS -- a file system manipulation subsystem which allows the user to create, delete, and modify file system catalogs, subcatalogs, and named files. The file space, not file content, is manipulated with ACCESS.

- Command Loader -- a default subsystem which is invoked whenever an unrecognized command is given, either at system-selection level or in line numbered build mode. The input is assumed to be the catalog/file description of an H* file to be loaded and executed or a series of responses for a command file (CRUN) application.

- FDUMP -- a remote-terminal, word-oriented file inspection and maintenance facility for files, regardless of their format. The files may have been generated in either batch, remote batch, or time sharing environments.

- File and Record Control Subroutines (TSS) -- provides File and Record Control subroutines needed for FORTRAN, ALGOL, and JOVIAL. These subroutines may also be used in COBOL or may be called directly by programs written in GMAP. These subroutines also provide automatic functions for dealing with the variety of file and device types available on the system. See the File and Record Control manual.

- HELP -- permits a terminal user to obtain a detailed explanation of any system error message.

- JOUT -- provides a means for inspecting output from batch jobs. The batch job could be a job submitted using the JRN command with a disposition code of J or JOUT, a remote terminal batch job (GRTS), or a job submitted at the central site.

- TRACE -- a powerful, conversational debug tool which permits a time sharing program to be executed in a controlled environment.

- LODS -- provides a debugging environment for a specified Time Sharing subsystem by loading the Debug Trace Package with the subsystem.

- LODT -- similar to LODS, except that the debugging environment is provided for a user program resident on an H* file.

- LODX -- allows the user to load and execute a program resident on an H* file.

- Media Conversion Program -- a batch-world program that may be run either at the central computer site or entered through a remote batch terminal. It generates a standard format, time sharing text file from a suitable card deck, or conversely, produces a card deck from such a file.

- CONVERT -- provides for the conversion of textual information between physical file formats, the reformatting of files, and the initiation of batch jobs.

- RBUG -- a conversational debug routine which can be used in conjunction with jobs submitted to the batch environment using the JRN command. RBUG has all the capabilities of the DEBUG routine of the batch world, permitting the user to monitor execution of the program, insert and remove breakpoints, and alter contents of memory locations and registers dynamically, all in an interactive manner.

- SABT -- retrieves specific locations of the ABRT file for printing at the user's terminal or optionally on the central site printer. When the system aborts the user's program, the memory storage area containing the program is written to the ABRT file.

- SCAN -- provides a means of examining output of a batch job from a time sharing terminal; the batch job may have been submitted using the JRN command through remote batch or as a central site job with the output placed into the file system.

The primary functions of the time sharing command language are as follows:

- Initiation of processing within a subsystem (e.g., LIST and JRN commands)

- Storage, retrieval, and purge of permanent files (e.g., SAVE and OLD commands)

- Request for operations on temporary time sharing files (e.g., NEW and SEQUENCE commands)

- Request for pertinent operating information (e.g., HELP and STATUS commands)

- Direction of flow of control within the subsystem (e.g., DONE and BYE commands)

In addition to the usual time sharing facilities at his disposal, the Time Sharing System user also has access to remote batch facilities. This capability is provided by a group of functionally interrelated subsystems called the Terminal/Batch Interface Facility.

The time sharing terminal user can perform the following operations:

- Access and modify a file of information created in the batch or remote batch dimension.

- Submit a job, such as a GMAP assembly and execution, to the batch dimension and inspect the output directly from a terminal.

- Establish conversational communication between a batch program and the user's terminal.

- Use an adjacent remote batch terminal as a high volume, hard copy output device, and, indirectly, as a high volume input device.

The basis for this communication between the several processing dimensions is the GCOS File System, which provides a common data base for all users of the system, and the common interface provided by GCOS. The file system provides automatic storage and retrieval of symbolically named permanent files on high capacity storage devices. These files are readily accessible in any processing mode. As a byproduct, the use of physical file volumes, such as card decks and tape reels, actually handled and stored by the user is considerably de-emphasized.

Considerable effort has been made to standardize error messages and comments for all subsystems in the Time Sharing System, and to have error message explanations immediately available at the terminal. Identical error or exception conditions arising in different subsystems are identified by the same error message text. Those messages that are not fully self-explanatory are prefixed with a message number enclosed by carets (i.e., <nn>), in almost all cases. This message number relates to a message explanation as given by the HELP subsystem. Upon encountering an error message that is not fully understood, the user can call the HELP subsystem and give the error message number when the number is requested, and receive an explanation of the error condition and suggestions as to possible courses of remedial action.

The Time Sharing System is completely modular and open-ended in that it is explicitly designed to allow user-implemented subsystems, tailored for a specific application, to be added to the Honeywell-supplied subsystems. This implementation of subsystems can be done readily, with no disturbance to the system. Specialized debugging facilities are provided for the checkout of new subsystems simultaneously with normal time sharing operation.

SECTION II

FILE SYSTEM

TEMPORARY USER FILES ASSIGNED BY TSS

The usage of standard temporary user files is described here on the basis of what is done by the Honeywell-supplied TSS subsystems, primarily BASIC and EDITOR. The designer of a new subsystem that requires a source file for each user may select this usage, both for overall system consistency and to take advantage of facilities already provided in TSS. All standard temporary files should have at least one asterisk (ASCII 052) in their names to differentiate them from user-created files.

There are two standard temporary files for each terminal user: the collector file, SY**, and the current file, *SRC.

Collector File (SY**)

The SY** file is automatically assigned to each terminal user by the TSS Executive. All terminal input except command language is collected on this file while the system is in build mode. This is the raw data received from the terminal. The collection of input is performed by the Line Service portion of the TSS Executive; that is, no subsystem is in execution. Thus, the assignment of SY**, the collection of input data on it, and the scanning of the input for command language are automatic functions of TSS, provided that the selected subsystem used build mode for the collection of new or additional input destined for a source file. Examples of SY** input are the numbered language statements in BASIC and the text entries in EDITOR.

Current File (*SRC)

The *SRC file receives the edited and/or merged version of the file with which the user is currently working. For example, if the user is writing a new BASIC program, the collector (SY**) file contains all the raw input, including any mistakes and corrections, other than keying errors corrected by commercial at sign (@) or CRTL/X.

When the user gives one of the BASIC commands, this causes the BSED subsystem to edit the data on SY** -- all corrections are applied, duplications removed, etc. SY** is then written to the current file, *SRC, which is the copy that is listed, run, or saved.

For an old BASIC program, the OLD file is copied directly to the user's *SRC file. Any changes that are typed are collected on SY** until a BASIC command is given. This causes the SY** file to be edited and then merged with the data on *SRC and the new, merged copy written to *SRC. Again, it is this new copy of the program that is run, listed, or saved.

In an OLD file, the user is always working with a copy of that file on *SRC -- either as is, or modified by SY** data -- and not the original. This feature leaves the OLD (permanent) file as backup copy (except when using OLDP/NEWP commands).

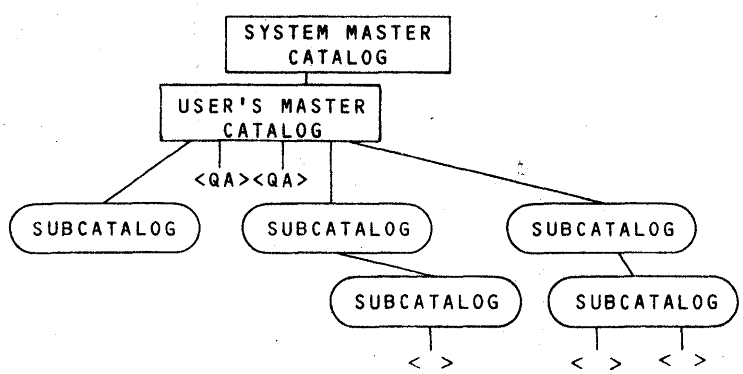
PERMANENT FILES ASSIGNED BY USER

TSS never assigns permanent file space to a user unless specifically told to do so by that user. Permanent files are handled by the File System, which is common to all programs operating under GCOS. Permanent time sharing files are ordinarily created by using SAVE or PERM commands; otherwise, they are created via the ACCESS subsystem or a batch FILSYS activity. (See also NEWP/OLDP commands.)

Structure Of The File System

The GCOS File System is described in the File Management Supervisor manual. The main points of interest to the TSS user are described below.

The GCOS File System is, in formal terms, a tree structure of indefinite length whose origin is the system master catalog. The primary nodes of the tree are user's master catalogs; the lower-level nodes are subcatalogs created by the user. The terminal points of the structure are the files themselves. (See Figure 2-1.)



Legend:

< > Denotes a file

<QA> Denotes a quick-access file

Figure 2-1. Logical Structure Of The File System

The master catalogs for each user are identified by USERID. A USERID must be unique within the system. All subcatalog and file names are automatically qualified by the user's master catalog name and the names of any intermediate subcatalogs. The system master catalog cannot be accessed by the normal user.

Catalogs And Files

A catalog consists of a description containing catalog name, password, and permissions. A catalog cannot be read or written, since it contains no user data.

In the GCOS File System, a file consists of a description containing file name, file size, password, permissions, and the specification of the physical file space. The file description is distinct from the physical file space, which may contain user data and can be read or written.

Passwords

Passwords can be attached to any catalog or file. A password simply allows a user to traverse a catalog/file string. The user can get to a given catalog or file only by giving the passwords for all higher-level catalogs in the string. The originator of a given string must also give the required passwords when traversing that string. However, when traversing a string, a password must not be given if none has been attached.

Permissions

Permissions, both general and specific, can be attached to any catalog or file. When permissions are attached at the catalog level, they apply to all subordinate catalogs and files. The originator of a catalog/file string has all permissions for that string but must give the passwords.

READ or R	Allow transfer of information from file to program but not from program to file.
WRITE or W	Allow transfer of information both from file to program and program to file.
APPEND or A	Not implemented. Treated as WRITE permission.
EXECUTE or E	Allow transfer of information from file to program but only for a compiler or loader. After the compiler or loader has completed its work, do not allow any transfer between program and file. Anyone with READ permission has EXECUTE permission. In any of the compilers, if the file is a source file and the only permission is execute, the RUN command cannot be used to build an object (H*) file.
RECOVERY or REC	Allow WRITE when file is abort locked or has defective space. Also accept directive to abort lock the file or to reset an existing abort lock. Anyone with RECOVERY permission is also given permission to WRITE and hence READ.

PURGE or P	Allow file to be deleted (file description to be deleted and file space to be returned with or without prior overwrite of space) or catalog to be deleted and all subordinate files to be deleted. Anyone permitted to PURGE can also perform any of the actions permitted by RECOVERY, including WRITE and hence READ.
CREATE or C	Allow catalogs and files to be entered as subordinate to this catalog.
LOCK or L	Allow directive to security lock the file or catalog (which security locks subordinate files) or to remove an existing security lock. A security lock does not apply to users with LOCK permission (since they are able to remove the lock).
MODIFY or M	Allow catalog or file description to be modified. Allow entries to be made in catalog for subordinate files or catalogs. Anyone permitted to MODIFY is allowed to perform any actions, since he could change permissions to give himself permission to perform these. Hence MODIFY includes CREATE, LOCK, and PURGE, which in turn includes RECOVERY and hence WRITE and READ.

Multiple concurrent reading or executing of a file is allowed by the File System, but multiple writing or appending is not.

User's Contact With The File System

The terminal user's contact with the GCOS File System is mainly through the Old-New (OLDN) and Save/Resave-Purge (SAVE) subsystems.

OLDN, when OLD is selected, writes the contents of the permanent (OLD) file onto the user's current file, *SRC. SAVE or RESAVE writes the contents of *SRC onto the named permanent file. In either case, to "access" a permanent file means to enter it into the user's Available File Table (AFT), as explained in the following description of the AFT.

AVAILABLE FILE TABLE (AFT) USAGE

TSS maintains an Available File Table (AFT) for each user. Before any I/O can be done on a file, an entry for that file must be placed in the AFT.

The AFT allows sufficient file descriptions to be kept in memory, thus minimizing the access time for these files. The AFT also allows files to be identified by their file names alone; for permanent files, the full file description may consist of many catalogs and passwords.

Temporary Files

DRL DEFIL (Define and Access a Temporary File) creates a temporary file and places the file entry in the AFT. Every temporary file defined by a subsystem should contain at least one special character (that is, other than alphabetic, numeric, period, or hyphen) in the file name. The asterisk is used by Honeywell-supplied subsystems. Since special characters are not allowed in permanent file names defined from a terminal, any conflict is avoided.

DRL RETFIL (Return a File) removes the file entry from the AFT and releases the file space back to the system. When a subsystem is finished with a file, it should return the file. All user's files in the AFT are released upon termination.

Permanent Files

DRL FILACT function number 4 (Access File) places the file entry in the AFT and sets the file "busy" for the permissions requested.

NOTE: This function does not create a file. Before a permanent file can be accessed it must have been created by DRL FILACT function number 3 (Create File).

DRL RETFIL (return a file) removes the file entry from the AFT and sets it "not busy" with respect to the current user. The file is not released from the file system when a DRL RETFIL is issued, but rather is detached from the current user's process (deaccessed).

FILE I/O

After the file is placed in the AFT, the following can be executed:

DRL DIO -- Reads or writes a file

DRL FILSP -- Positions a file forward or backward

DRL REW -- Positions a file to its beginning

DRL MORLNK -- Increases the size of a temporary file

DRL GROW -- Adds space to a permanent file, up to its maximum size

DRL PART -- Releases a portion of a temporary file

DRL SWITCH -- Switches two temporary file names

These are the only file I/O derails that affect or relate to the AFT and are most often used by subsystem programs. The others (all of the FILACT functions except Access File) affect only the GCOS File System.

SECTION III

TERMINAL USAGE

This section contains general descriptions of and operational procedures for several remote terminals. For complete details pertaining to a particular terminal, the user should refer to the instruction manual accompanying the terminal unit.

TELEPRINTER OPERATION

Terminal Applications

The following types of teleprinter terminals, or their equivalent, may be used to communicate with the Time Sharing System:

- IBM 2741
- Teletype models 33, 35 and 37
- GE TermiNet 300

These terminals communicate with the Time Sharing System via the Remote Terminal Supervisor (GRTS) or the Network Processing Supervisor (NPS). These interfaces are described in the Remote Terminal Supervisor (GRTS) manual and the Network Processing Supervisor (NPS) manual.

Each time a key is struck, the character is transmitted to the remote communication system and stored until the carriage return is struck. A carriage return indicates that the line is complete. The number of characters in a line may range up to 160 characters plus a carriage return.

If the terminal is equipped with a paper tape reader/punch, this device may be used for input/output. The input must be formatted the same as for keyboard input, but each line must be terminated with carriage return, line feed, and two rubouts. The input tape must be terminated with an ASCII X-OFF (or DC3) character.

Editing

Keyboard input is sent to the host computer in units of complete lines. A line of terminal input is terminated by a carriage return. Therefore, corrections to a line-in-progress (i.e., a partial line not yet terminated) can be made.

A typing error detected before the line is terminated can be corrected in one of two ways. One or more characters may be deleted from the end of a partial line or the incomplete line and may be cancelled. Character or line deletions are effected by means of two special characters designated as control characters. These control characters may differ between terminals.

For teleprinter terminals

character	control function
@(commercial at sign)	character deletion
CTRL plus X keys	line deletion

For IBM 2741 or DATEL terminals

character	control function
1/4 (or degree symbol)	character deletion
$\underline{+}$	line deletion

NOTE: Line deletion does not occur until a carriage return is given or ATTN (IBM 2741) or INT (DATEL) is pressed.

The editing rules are as follows:

o Use of the character-delete control deletes from the line the character preceding the deletion character; use of _n_ consecutive deletion characters deletes _n_ preceding characters (including blanks) up to the beginning of the line.

For example:

*ABCD<u>F</u>@E would result in ABCDE being transmitted to the program file.

*ABC<u>bDEF</u>@@@@DEF would result in ABCDEF being transmitted.

(The characters to be deleted are underlined for illustration.)

o Use of the line-delete control causes all of a line to be deleted. The characters DEL are printed to indicate deletion. For example:

```
*ACDEFG CTRL/X  DEL        (all characters deleted;
                            carriage return automatic)

                           -(ready for new input)

                    or

*ACDEFG+ (carriage return)

DEL                        (all characters deleted)

                           -(ready for new input)
```

NOTE: CTRL/X, ATTN, or INT do not require a carriage return.

The control-character pair for each type of terminal cannot be used for other than the deletion function assigned them.

In AUTOX and AUTO (automatic line number generation) line numbers and spaces are not deleted.

Logon Procedure

To initiate communication with the Time Sharing System, the user performs the following steps:

- o Turns on the terminal
- o Obtains a dial-tone on the associated phone-set
- o Dials one of the numbers of his time sharing center

The user will then receive either a busy signal to indicate that the line is not presently available or a high-pitched tone -- a "beep" -- to indicate that his terminal has been connected to the computer.

The Time Sharing System is then prepared to output a logon message; either automatically (no terminal action required) or following a carriage return from the terminal. The following is a sample of the automatic logon message:

HIS TIMESHARING ON date AT time CHANNEL nnnn TS1

where time is given in hours and thousandths of hours (hh.hhh), and nnnn is the user's channel number. This is the standard message, however the user site may put in a message of its own.

The following is a sample of the logon message when a carriage return is required:

110601
HIS TIMESHARING ON date AT time CHANNEL nnnn TS1

The number "110601" identifies the type of channel to which the terminal is connected. For a detailed explanation of the meaning of this number, refer to the Remote Terminal Supervisor (GRTS) manual or the Network Processing Supervisor (NPS) manual.

Following this message, the system asks for the user's identification:

USER ID -

The user responds, on the same line, with the user-ID assigned by the time sharing installation management. This user-ID uniquely identifies a particular user already known to the system. This ID is used to locate his programs and files subordinate to the SMC and to account for usage of the time sharing resources. An example request and response might be:

USER ID -J.P.JONES

NOTE: User's responses are underlined for illustrative purposes.

A carriage return must be given following any complete response, command, or line of information typed by the user. If a charge number is also required for accounting purposes, the user can supply it as follows:

USER ID -J.P.JONES;1234567E

The charge number may consist of from 1 to 12 alphanumeric characters, separated from the user-ID by a semicolon.

After the user responds with his user-ID, the system asks for the sign-on password that was assigned to the user along with the user-ID as follows:

PASSWORD--

The user should type the password directly on the "strikeover" mask provided below the PASSWORD request. The password is used by the system as a check on the legitimacy of the named user. If either the user-ID or password is given incorrectly two consecutive times, the user's terminal is immediately disconnected from the system.

On teletype-compatible devices, after the password is entered on the strikeover mask, a random alphabetic character string is typed over the password entry. The user's password is thus "sandwiched" between strikeovers for hard copy devices or totally overwritten for screen displayed devices.

On Visual Information Projection (VIP) devices, upon receipt of the password, a Reverse Line Feed (RLF) character, followed by a string of spaces, is issued to erase the entry from the screen.

The user-ID and password may be given on the same line when the query "USER ID-" is issued, separated from one another by a dollar sign ($). Charge number, when specified, must follow the password and be separated from the latter with a semicolon. Note that security is compromised by entering the password in this manner, since it is not typed on a strikeover mask. Assuming the password of user J.P. Jones is "JPJ", this method of logon would be:

USER ID - J.P.JONES$JPJ;1234567E

The system attempts to overstrike the user's password after it is entered in conjunction with the user-ID. The overstrike utilizes a backspace (BSP) character that may not be recognized by some terminal types. Any line (CTL/X) or character editing (@) use during entry of the combined user-ID and password also effects the completeness of the password overstrike.

At this point, if the accumulated charges for the user's past time sharing usage equals or exceeds 100 percent of the current resource allocation, a warning message is sent:

RESOURCES OVERDRAWN n%

If the accumulated charges exceeds 110 percent of the current resources, the user receives the following message and is immediately disconnected:

RESOURCES EXHAUSTED - CANNOT ACCEPT YOU

If the user's file space utilization (used:available) is greater than 88 percent used, the following information message is sent:

n BLOCKS FILE SPACE AVAILABLE

The number n specifies the number of 320-word blocks of unused file space for this user. This does not affect the logon procedure, and the user is permitted to continue.

The following is an example of a complete logon procedure, up to the point where the user is ready to begin file building or exercising commands:

```
HIS TIMESHARING ON 05/26/77 AT 14.568 CHANNEL 0012

USER ID -J.P.JONES
PASSWORD
XXXXXXXXXXXX
*              - (the user begins entering input on this line)
```

The BRN, FRN, and JRN commands can be issued independent of previous system selection (if any) and imply RUN for BASIC, FORTRAN, and batch job submission, respectively. The command BSEQUENCE can be used to resequence a BASIC file and SEQUENCE can be used to resequence a non-BASIC file, independent of the current system selection.

Entering Build Mode Input

Following the logon procedure, the user is in build mode (as indicated by the initial asterisk) and is ready to build files and/or exercise commands. All lines of input other than commands are accumulated on the user's current file. This is normally the file that contains the program or text the user wants to work with. If the user is building a new file, the current file will initially be empty.

If the user has recalled an old file (OLD filename) the content of the named old file will initially be on the current file. Any input (except control commands) will either be added to, merged with, or replace lines in the current file, depending upon the relative line numbering of the lines in the file and the new input. (Refer to "Correction or Modification of Line-Numbered Files" below.)

Following each line of input (that is not a command) and terminating carriage return, the subsystem supplies an initial asterisk, indicating that it is ready to accept more input. In the case of command language input, the user is normally returned to build mode following execution of the process requested by the command.

A line of file building input must begin with a line number contained within the first eight character positions of the line. This number may optionally be preceded by one or more initial blanks. The line number facilitates correction and modification of the source program. The line number is always terminated, (i.e., immediately followed) by a nonnumeric character, which may be a blank.

Correction Or Modification Of Line-Numbered Files

The correction or modification of the current file in line number sequence proceeds according to the following rules:

- o Replacement: a numbered line will replace any identically numbered line that was previously typed or already contained on the current file; i.e., the last entered line numbered nnn will be the only line numbered nnn in the file.
- o Deletion: a line consisting of a line number only, (i.e., nnn), will cause the deletion of any identically numbered line that was previously typed or is already contained on the current file.
- o Insertion: a line with a line number value that falls between the line number values of two existing lines will be inserted in the file between those two lines.

At any point in the process of entering file building input in line-numbered subsystems, the LIST command may be given, which results in a clean, up-to-date copy of the current file being printed. In this way, the results of any previous corrections or modifications can be verified visually. Following the command OLD filename, the LIST command can be used initially to inspect the contents of the current source file; i.e., the "old" program.

Automatic Terminal Disconnections

Once communication with the Time Sharing System has been established, any question or request must be answered within ten minutes. If these time limits are exceeded, the user's terminal will be disconnected. A time-out of the user's line will occur when the line has I/O pending (e.g., output then input request).

Logoff Procedure

To terminate the user's current session with the Time Sharing System and disconnect the terminal, the BYE or LOGOFF command may be given.

```
*BYE

 or

*LOGOFF
```

A report of the user's time sharing usage charges is given, as illustrated by the following example, and the terminal is disconnected.

```
**COST:  $     0.17 TO DATE:  $  206.11=21%
**ON AT 15.000 - OFF AT 15.016 ON  04/19/77
```

If the BYE command is used, prior to the issuance of the user's usage charges, the AFT is scanned for the user's temporary files and the user is queried as to their disposition if such files exist. A carriage return in response to temporary file disposition will release the temporary file(s) listed. To save one or more of the listed files, the user responds with the file names to be saved.

To terminate the current session without disconnecting the terminal, the command NEWUSER may be given in place of BYE. This procedure allows another user to logon immediately. The current user's logoff report is then printed and a new logon sequence is initiated. NEWUSER may also be used to change the charge number, but without going through the logoff/logon procedure.

Failure to follow logoff procedures as described above may result in unpredictable problems (lines or files remaining busy, etc.). Certain data sets do not automatically disconnect after logoff from the terminal. In such cases, it is necessary to manually disconnect the data set by lifting the handset, pressing the talk button, and hanging up the handset when the dial tone is heard.

Terminating An Output Process

A lengthy listing or other output of information at the terminal, initiated for example by a LIST command, may be prematurely terminated by the use of the interrupt control peculiar to the type of terminal in use. This interrupt control is as follows:

- For teleprinter terminals -- the BREAK key
- For typewriter-like terminals -- the ATTN or INT key

This control can also be used for abnormal termination of a program execution. However, the user is cautioned against indiscriminate use of this control since the results of its use are in some cases unpredictable (in regard to the status of files, for example). The subsystem will normally return to build mode or to the subsystem selection level following the use of an interrupt control.

Paper Tape Input In Build Mode

In order to supply file-building input from paper tape, the user gives the command TAPE (#TAP if the subsystem is Text Editor). The subsystem responds with READY. If the tape reader is ready, it will be turned on automatically. If it is not ready, the user should position his tape in the tape reader and start the device. Input is terminated when an X-OFF (or DC3) character is read by the paper tape reader, or the tape is stopped and the user types X-OFF (or DC3).

The tape may be prepared offline from the keyboard, or it may be the result of previous output punched by the paper tape unit. If prepared offline, it should include carriage returns to terminate each line, just as if entering data online, plus explicit line feeds to obtain legibility on the terminal printer during preparation and transmission. The carriage return and line feed must be followed by two rubout characters for terminal timing considerations.

Command language may not be included on the tape. The input should be preceded by several rubout characters and terminated by an X-OFF (or DC3) followed by several rubout characters. Neither the X-OFF (or DC3) nor the rubout characters will appear in the file.

As with keyboard input, a maximum of 160 characters is permitted per line of paper tape input. Excessive lines will be truncated at 160 characters, with the remaining data placed in the next line. A maximum of two disk links (7680 words) of paper tape input will be collected during a single input procedure, except in LUCID mode, which has a limit of six links. All data in excess of two disk links will be lost.

Building File From Non-ASCII Paper Tape

In order to supply file building input from non-ASCII paper tape (unaltered eight-bit codes), the user gives the command LUCID instead of TAPE. The system reads in the tape and stores the data on a file without editing or parity modifications. The system does not delete or act on any characters in the data stream, such as DEL, X-OFF (or DC3), CR, etc. The input will be terminated when a pause of over one second occurs in the data transmission. Termination does not require an X-OFF (or DC3) character, as does normal paper tape input via a Front-End Network Processor.

NOTE: LUCID cannot be used if data communication is via a Low-Speed Line Adapter (LSLA) or an Asynchronous Communication Base (ACB) on a DATANET 355/6600 Front-end Network Processor.

During paper tape input via a Front-end Network Processor, the paper tape input will stop when an error message is to be sent to the terminal.

Automatic Paper Tape Input

At any point during the operation of the Time Sharing System and at a time when the user must supply keyboard input, a previously prepared paper tape in special format may be used to simulate a sequence of responses, one line at a time. The system need not be in build mode and direct (i.e., conversational) responses, file building input, and/or commands may be entered.

This feature allows the preparation of a paper tape for input to the Time Sharing System and/or subsystem(s) prior to connection with the system and allows terminal operation without supervision during the connection. The paper tape input may be for a specific subsystem or production program execution only, or may include anything from logon through logoff procedures. Obviously such a tape must be error free.

The required format for each input line is as follows:

data string (up to 80 characters)
carriage return
X-OFF (or DC3)
RUBOUT (may be multiple, but one is minimum requirement)

Character-delete control characters may be used. Line-delete controls must be used as follows:

data string (to be deleted)
(line-delete control) character
X-OFF (or DC3)
RUBOUT (one is minimum)
corrected data string
carriage return
X-OFF (or DC3)
RUBOUT

NOTE: Parity errors encountered during paper tape input may cause the terminal to be disconnected.

It is suggested that extraneous line feeds not be included in the tape. If, however, the user desires line feeds for terminal printer legibility, they should be either between the data string and carriage return, or one line feed immediately following X-OFF (or DC3).

To initiate automatic paper tape input, the user should position the tape and start the reader at any time that keyboard input is required.

The terminal is automatically disconnected if no input is received within ten minutes of the request for such input, whether via paper tape or keyboard.

KEYBOARD/DISPLAY TERMINAL OPERATION

The keyboard/display terminals are cathode-ray tube display devices which are similar in operation to the teleprinter terminals. This section describes operation of some types of display devices commonly used with the Time Sharing system:

- o DATANET 760 VIP (Visual Information Projection)
- o 765/775/785'7700 Series VIP

The keyboard for the 775/785 Series VIP is shown in Figure 3-1. Most of the display devices have a similar keyboard. Some of the keys and their function are discussed here, but for a complete description, the user should refer to the manual for the specific device.

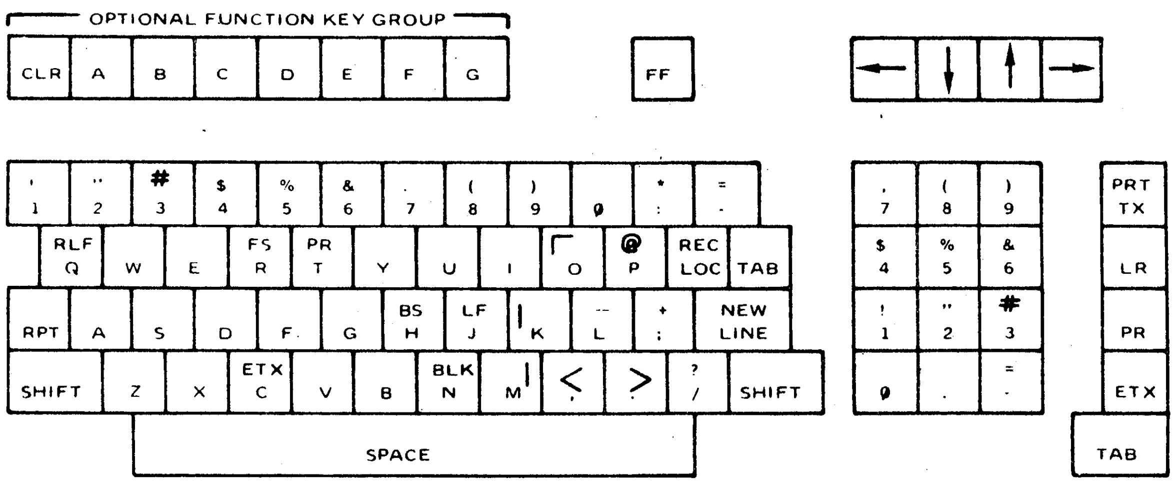


Figure 3-1. Keyboard For 775/785 Series VIP Keyboard

The Time Sharing System can interface with most of the VIP terminals. Both synchronous and asynchronous units are available, with line speeds of 1200, 2000, 2400 or 4800 bits per second. A complete page of input may be composed before transmission to the Time Sharing System, and a complete page of output may be displayed. The page consists of 4 to 26 lines, depending on the model. (See Table 3-1.)

NOTE: The number of characters transmitted or received is subject to limitations of the terminal. Also the user should reference the Remote Terminal Supervisor (GRTS) manual.

Table 3-1. Display Devices

Terminal Type	Device Code	Char/ Line	Lines/ Page	Char/ Page	End of Text Symbol	Transmit Symbol	Print Symbol	Receive Symbol
765/775 VIP	11	46	22	1012	‖	none	none	none
785/786 VIP	12	92	22	2024	‖	none	none	none
7700 VIP	13	80	12	960	‖	none	none	none
7700 VIP	14	46	22	1012	‖	none	none	none
7700 VIP	15	80	24	1920	‖	none	none	none
7800 VIP	22	80	24	1920	‖	none	none	none

The keyboard/display terminals differ significantly from the keyboard/printer terminals in entering data. As mentioned above a complete page can be entered by one transmission; also while the user is composing the input from the keyboard, the terminal is in effect offline since no data is transmitted until the user initiates the proper transmit procedure.

CAUTION: The system automatically disconnects any terminal which does not input (transmit from the terminal) within ten minutes.

Also when a user requests output (e.g., LIST), only a full page is sent even though the file could be longer. The remainder of a file may be displayed, a page at a time, by continued requests for transmissions until the end-of-file is reached.

Data Display And Transmission

The keyboard is similar to that used with hard-copy terminals. Most of the keys are in the same physical location on the keyboard. Although the user should depend upon the instruction manual accompanying the unit for the function of special keys, some of those special keys are discussed here.

After the unit is turned on and allowed time to warm up, an entry marker should appear in the upper left-hand corner of display unit. This entry marker is the position on the display where the next character or space will be entered. As a key or space is struck the entry marker advances to the next position. When the end of a line is reached, the entry marker moves to first character position (left side) of the next line. When the end of the last line on the page is reached, the entry marker moves to the top of the screen, first character position.

NOTE: The entry marker can be positioned by the use of special keys without changing or clearing the display. The most obvious are the four arrow keys. Some devices (see Figure 3-1) have line return (LR), page return (PR), new line, backspace (BS), and forward space (FS) keys which also position the entry marker without changing the display.

The entry marker also marks the point where transmission to the computer is to begin. For example to enter one line, possibly a one-word command, the steps are as follows:

1. Type the word -- LIST
   The entry marker now appears at the space following the "T" in LIST.
2. Press ETX, end of text. The ETX symbol is two vertical lines (||) or C.
3. The entry marker moves another space and must be moved to the first letter to be sent, "L" in this case. LR line return will return the marker to the beginning of the line or with the backspace key (←).
4. Pressing TX (transmit) sends the line, LIST, to the system.

On some keyboard/display terminals the transmit sequence (ETX, LR, TX) is generated by pressing a single function key.

In general, a transmission is bracketed by the position of the entry marker, and end-of-text which may be one or more lines.

Logon

After turning on the unit, the user should allow time for it to warm up. Some units require approximately 30 seconds to warm up. The entry marker should appear on the screen before continuing.

When the keyboard/display unit is ready, the user dials the number of the time sharing center. The following is a typical logon procedure (user responses are underlined; comments in parentheses):

```
$*$SC PASSWD,NN,TSS
(screen is cleared by the system)
112501
HIS TIMESHARING ON DATE AT TIME CHANNEL NNNN   TS1

USER ID -
UR-IDENT
PASSWORD - -
                      (password erased from screen)
OLD FILEX
(screen is cleared by the system)
```

Where: SC - user selected station code

PASSWD - password

NN - number of lines per page: 04, 08, 12, 16, 22, 22L, 22N, 24 or 26

The initial logon input:

$*$SC PASSWD,NN,TSS

is a requirement of the particular system configuration and may vary. Users should be notified by their computer operations group as to the exact format required at the site. The message:

TERMINAL DISCONNECT   ISP

is displayed to notify the user that the logon was incorrectly entered. Since the terminal is still online, the user may attempt to enter a corrected logon message. It is not necessary to re-dial.

Additional rejection error codes may be sent to the terminal as a result of some central system detected error condition. (See Remote Terminal Supervisor (GRTS) manual).

Logoff

The logoff procedure and the logoff message are identical to that for the teleprinter. However, in addition to the BYE command the display units may be disconnected with the command "$*$DIS".

Unique Features

In addition to the logon procedure, the following features are unique to the keyboard/display terminals.

- TAPE/#TAPE/LUCID/#LUCID commands cannot be used.

- More than one line of data may be transmitted to or received from the system at one time.

- Special character-delete and line-delete control characters are not applicable, as all errors may be corrected by positioning the entry marker over the erroneous character and typing the correct one.

- A LIST or other output commands will display only one page, up to 2024 characters. If a file is longer than one page, the remainder of the file may be displayed by either depressing the print (PRT) key or repeating the output command.

- A "BREAK" (interrupt) signal is transmitted to the system by means of the following control message:

  $*$BRK

  This message can be used to interrupt some lengthy output process, such as the unwanted remainder of a long listing, or to interrupt execution of a user's program.

- With the use of algebraic subsystems BASIC and ABACUS, the up-arrow (↑) symbol used as the exponentiation operator is replaced by a BLK (blink) character preceding the exponent. The blink character itself is displayed as a blank, and causes the exponent character(s) following the blank, in turn, to blink.

7700 SERIES VIP TAPE CASSETTE AND PRINT PAGE ADAPTER OPERATIONS

The 7700 Series VIP can read or write to a tape cassette unit or direct data to a print page adapter for hard-copy printing. All three units are capable of offline operation. For a description of offline operation, refer to the <u>7700 Series Visual Information Projection (VIP) Systems</u> manual, Order Number AL29.

Output To Cassette

Cassette output is initiated with a WRITE TAPE n command where n is the tape number (n=1 or 2, default tape number =1). The Time Sharing System directs output to the designated tape cassette unit. The output is also displayed on the screen. The WRITE CEOF n command disables the cassette output mode of operation and writes an end-of-file (EOF) on the current tape.

Example:

```
*OLD TEST
*WRITE TAPE 1
*LIST
10   THIS IS A TEST         Displayed on screen and
20   OF THE TAPE            sent to tape cassette.
30   CASSETTE FUNCTIONS
40   END
*WRITE CEOF 1               Writes EOF on tape.
EOF
*
```

Input From Cassette

Cassette input is initiated with a READ TAPE n command which is similar to the TAPE command used to initiate paper tape input. Data from the cassette is transmitted in variable sized blocks of up to a maximum block size equal to the screen size. The cassette tape must have an EOF (written on the tape with the WRITE CEOF n command) to terminate the TAPE READ sequence.

Example:

```
*NEW
*READ TAPE 1
10   THIS IS A TEST              Sent from cassette
20   OF THE TAPE                 to screen and to
30   CASSETTE FUNCTIONS          the system.
40   END
EOF
*
```

Echo Back

The 7700 terminal operator can have the data that is keyed in recorded on the print page adapter as well as on the system by depressing the PRINT KEY. The Time Sharing Executive "echoes" the input block back to the printer.

Backspace Cassette

The BSP TAPE n command enables the terminal user to backspace the cassette tape one record.

Rewind Cassette

The REW TAPE n command enables the terminal user to rewind the cassette tape.

Output To Printer

The PTON command enables subsystem output to be routed to the print page adapter. The PTOF disables the printer mode of operation in TSS.

Example:

```
*OLD TEST
*PTON
*LIST
                    All output goes to printer - not displayed
                    on the screen.
*PTOF
*
```

or

```
*NEW
*PTON
*CATALOG
                    Catalog output goes to printer - not
                    displayed on the screen.
*PTOF
*
```

Continuous Output Mode

When data is routed to the cassette unit or to the printer, a continuous stream of data is transmitted. While the data is visible on the screen, for cassette functions, the blinking asterisk is omitted and there is no need to press the print key for each page.

Summary Of 7700 Cassette/Printer Commands

PTON	print on
PTOF	print off
WRITE TAPE n	write cassette n
WRITE CEOF n	write EOF on cassette n
READ TAPE n	read cassette n
BSP TAPE n	backspace cassette n one record
REW TAPE n	rewind cassette n

Where: n is optional; if not specified, the default value is 1. If specified, n is either 1 or 2.

Caution: These commands can only be used on 7700 Series VIP (device 13, 14, or 15 octal).

COMMANDS FOR VIP TERMINALS

In addition to the cassette/printer commands, the commands described below are only for VIP-type terminals.

Form Feed Commands

The no form feed command (NFORM) allows the user to control the transmission (from the Time Sharing Executive) of the form feed character after the prompt for a page request. The NFORM command can be entered at system selection level or at build input level. It causes the cursor to be returned to column one - in character position one of the screen. This leaves all of the previous data on the screen. The command FORM reverses the NFORM command, whereby form feeds are again transmitted after the prompt for a page request.

Case Commands

These commands are for VIP terminals which can display both uppercase and lowercase characters. At logon all lowercase characters are transliterated to uppercase characters. The command LCASE allows both uppercase and lowercase characters to be transmitted to the VIP terminal. The command UCASE reverses the operation.

SECTION IV

COMMAND LANGUAGE REFERENCE

The time sharing user accomplishes tasks during a session by entering commands or service requests in response to a system prompt. The full complement of supplied commands available to a user is described in this section. Each command is described in the following sequence:

Purpose	describes the function of the command
Format	describes the full capabilities of the command in a standard notation. The standard notation is described below under "Command General Form".
Description	describes in more detail some of the parameters used in the general form. If the command requires no arguments or requires rather straight forward options, this section may be absent.
Discussion	the function of the command describes any special uses and notes any limitations of which the user should be aware.
Examples	shows different uses of the command.

Command General Form

The general form of each command is described using a standard notations. The following is a complete set of rules for the notations used in the general form:

1. Material enclosed in square brackets, [ ], indicates optional parameters. They may be included or omitted as required by the user.

2. When material is enclosed in braces, { }, one, and only one, of the enclosed parameters must be chosen.

3. An ellipsis, ... , indicates that the preceding parameter may be repeated.

4. Material enclosed in angle brackets, < >, represents parameters the user is to supply.

5. The vertical bar "|" indicates that a choice among the parameters separated by vertical bars must be made.

6. All text printed entirely in capital letters must be typed as is, unless the portion of the general form containing it is itself optional.

7. All underlined notation is required, unless the portion of the format containing it is itself optional. Any underlined portion overrides any implied editorial notation (i.e., [ ] { } < > |).

8. Any usage of spaces, semicolons, commas, dashes, parentheses, etc., in the general form indicates required punctuation.

9. ::= may be interpreted as "is defined to be". Parameters enclosed in angle brackets, < > , are separated from their definition by the ::= symbol.

For example, using this notation, the general form of the RELEASE command is described as follows:

General Form   RELE[ASE][<filedesc>[;<filedesc>]...]

In the above example, special meaning is implied by the notation. Since square brackets indicate that the enclosed items are optional, RELE[ASE] means that ASE is optional in typing the command name, and [<filedesc>[;<filedesc>]...] means that this field, <filedesc>[;<filedesc>]..., is optional after the command name. Within this latter field is another optional field, namely, ;<filedesc>. The ellipsis that follows ;<filedesc> indicates that this field may be repeated. The angular brackets that appear in the field, <filedesc>, indicate that the user supplies a name for that field. Depending upon which optional parts are used, the punctuation that may be required is a space and, possibly, semicolons. In summary, the RELEASE command may be typed as RELE or RELEASE followed by none, one, or more than one file name. The command name and the first file name must be separated by a space; any additional file names must be preceded by semicolons.

Some of the valid forms for the RELEASE command are:

1)	RELEASE	(prompts for the name of the file to be released)
2)	RELE	(same as RELEASE)
3)	RELEASE FILEA	(releases the file FILEA)
4)	RELE FILEA FILEB	(releases the files FILEA and FILEB)
5)	RELE FILEA;FILEB;FILEC	(releases the files FILEA, FILEB, and FILEC)

The usefulness of the notation becomes clearer when more difficult commands are considered.

Example 1

Format

```
OLD[ <file-ref>[{;|:|#}<file-ref>]...]

    <file-ref>   ::- {*|<filedesc>}[(<line-range>)]
    <line-range>::= <begin-line>-<end-line>
                  | <begin-line>-
                  | -<end-line>-
    <begin-line>::= <line>
    <end-line>  ::= <line>
    <line>      ::= a 1- to 8-digit decimal number
```

Examples

1) OLD

(prompts for the file name)

2) OLD DATA74

(the contents of file DATA74 replace the prior contents of the current file)

3) OLD MAIN;SUB1;SUB2

(the contents of files MAIN, SUB1 and SUB2 are concatenated in the order listed and replace the prior contents of the current file)

4) OLD WEIGHTS(100-500);VOLUME(100-500)

(lines 100 through 500 of file VOLUME are appended to lines 100 through 500 of file WEIGHTS and they replace the prior contents of the current file)

5) OLD PROG1;FIXES;SUB1;SUB2*ALTERS

(merges the contents of the files PROG1 and FIXES, concatenates with the result the contents of files SUB1 and SUB2, and merges the contents of the file ALTERS with the result)

Example 2

Format

```
LIST[H|E[<columns>]][ [<file-list>:<line-list>]]

or    LISTL[ *:<filedesc>]

 <columns      ::= a decimal number
 <file-list>   ::= <file-ref>[;<file-ref>]...
 <file-ref>    ::= [*|<filedesc>}[(<line-list>)]
 <line-list>   ::= <line-ref>[,<line-ref>]...
 <line-ref>    ::= <line>|<line-range>
 <line-range>  ::= <begin-line>-<end-line>
                 |  <begin-line>-
                 |  -<end-line>
 <begin-line>  ::= <line>
 <end-line>    ::= <line>
 <line>        ::= a 1- to 8-digit decimal number
```

Examples

1) LIST
   (lists the entire contents of the current file)

2) LIST 50-80,90,120
   (lists lines 50 through 80, 90, and 120 of the current file)

3) LIST CLARION(10-50,100-150,);TEMPIN(10,30,220-)
   (lists lines 10 through 50 and lines 100 through 150 in file CLARION; and, lists lines 10, 30, and 200 through the end of the file TEMPIN)

4) LISTH TESTFILE
   (lists the entire contents of file TESTFILE with a date and time header)

5) LISTE /STUDENT/GRADES
   (lists the entire contents of the file GRADES in subcatalog STUDENT with each line in the file listed with 72 columns per line on the terminal)

6) LISTL
   (lists the last line of the current file)

Generally the user will be able to use a command after examining the examples and reading the comments for a command. But, to understand the full capabilities of a command, the user must understand the general form.

Purpose

The ABC command invokes the ABACUS subsystem to evaluate a user supplied arithmetic expression.

Format

```
ABC [ <expression>]

  <expression>::= arithmetic expression to be evaluated
                  FOR variable - IV, LV, STEP; arithmetic expression
          <IV>::= initial value for FOR variable
          <LV>::= limiting value for FOR variable
        <STEP>::= step increment for FOR variable.
                   A step of 1 is assumed.
```

Discussion

The ABACUS subsystem evaluates arithmetic expressions in an interpretive manner and displays the results. Using expressions or iterative FOR loops, the subsystem can calculate the answers to a variety of mathematical problems.

The subsystem can carry intermediate results forward into other calculations by assigning the interim results to named variables. Variable names are composed of one to eight alphanumeric characters, the first of which must be alphabetic: e.g. A1, B, SQRTSUM. Variable names are used within ABACUS to perform an assignment of value and are of the form "variable = expression". Variables can appear as part of free-standing expressions or in FOR loop assignments. When a variable is used within the scope of a FOR loop it no longer is preserved for follow-on calculations.

Expressions in ABACUS can employ a mixture of integers, decimal fractions, or numbers expressed in scientific notation. Using the standard arithmetic operators for addition, subtraction, multiplication and division augmented with a summation operator (& - ampersand) and an exponentiation operator (↑-up arrow), the user can construct a variety of expressions. Parenthesis may be used freely to clarify the evaluation of an expression.

ABACUS has a number of constants and functions which may be used during the construction and evaluation of expressions:

CONSTANT	DEFINED VALUE
PI	3.14159...
RADIAN	57.295...
E	2.71828...
	significance to 18 digits

FUNCTION	MEANING
ABS(X)	Absolute Value of X
ATN(X)	Arctangent of X
COS(X)	Cosine of X
EXP(X)	e to power of X
LOG(X)	Natural logarithm of X
SIN(X)	Sine of X
SQR/SQT(X)	Square root of X
TAN(X)	Tangent of X
INT(X)	Integer of X
	For trigonometric functions X is an angle in radians

Function names are reserved words and may not be used as variable names. Constant names may be used in a variable assignment if the constant value is to be changed.

To compute a summation, the ampersand must appear at the beginning of an expression and may not be enclosed in parenthesis. The entire portion of the expression is assumed to be the argument to be summed. Examples of FOR loop usage may be combined with summation expressions.

If the step value, c, is not specified, 1 is assumed. Substitutions for a, b, and c may be positive or negative integers, expressions, or predefined variables.

For example:

```
? FOR X = 1, 5;   FOR Y = 7, 50, 9;   Z = &(X+Y)*PI
Z = 2199.1149
```

In summations, all FOR variables are treated as summation indices and in the case of summations over two or three FOR variables, the indicated summations are nested. Each summation variable takes on the values a, a+c, a+2c,...up to but not exceeding the value b. Thus the expression above would expand as follows:

$$Z = \sum_{X=1,2,\ldots}^{5} \sum_{Y=7,16,\ldots}^{43} (X+Y)$$

$$Z = ((1+7)\,\pi + (1+16)\,\pi + (1+25)\,\pi + \ldots + (5+34)\,\pi + (5+43)\,\pi)$$

Although an expression containing a summation operator must be preceded by one or more FOR specifications (in order to be meaningful), FOR variables may also be used in expressions that do not contain the & operator. For example:

```
?  FOR A = 3, 11, 2;   FOR B = 1, 3;   X = A↑B
```

In these cases, the expression will be evaluated separately for each possible combination of FOR values (as is done in FORTRAN). The output from the example expression just above would appear as:

A	B	X
3	1	3
3	2	9
3	3	27
5	1	5
5	2	25
5	3	125
7	1	7
7	2	49
7	3	343
9	1	9
9	2	81
9	3	729
11	1	11
11	2	121
11	3	1331

If a label variable is used, as in the above example (X), the last determined value is remembered for the variable.

All calculations in ABACUS are performed in double-precision floating-point with a precision of 18 digits. Displayed results are limited to seven places in the fractional portion although 18 significant digits are carried internally.

Examples

```
1)    ABC              (prompts for the expression)
2)    ABC 3.14159*7.63-2.513
3)    ABC 1.379-2
4)    ABC  X = SIN (30/RADIAN)
5)    ABC  X = 5
           X 2*4
6)    ABC  Y = -5*2.41E-3
7)    ABC  X = 3
           Y = 3*(X)
           Z = 3(X)  3*(X), 3*X, and 3(X) are equivalent
8)    ABC
          ?X= 4*3*2/5
               X      =              4.8
          ?Y= SIN(45)
               Y      =        0.85090352
          ?Z= X*Y
               Z       =        4.0843369
9)    ABC FOR I=1,5;  A=Z*I
                i
                          1          a
                                     4.0843369
                          2          8.1686738
                          3          12.253011
                          4          16.337348
                          5          20.421685
10)   *ABC X=PI*5
          X               =          15.707963
```

Purpose

The ACCE command provides an interface to the file system which permits the manipulation of catalogs, files and the attributes of each. Subordinate to the user's own SMC/UMC entry, the ACCE subsystem allows the user to perform the following: create and modify structure, add or remove passwords, give specific or general permissions to catalogs or files, set file modes of access, create files and catalogs and alter file sizes, list structure, rename catalogs or files, and release structure.

Format

```
ACCE  <arguments>
ACCE  [user supplied short-form arguments][;args]

   where <arguments> is defined as:

     <function>,<pathname>,<option>....<option>

<function> ::= CC|CF|AF|DF|MC|MF|PC|PF|RC|RF|LC|LS
     <CC> ::= catalog create function
     <CF> ::= create file function
     <AF> ::= access file function
     <DF> ::= deaccess file function
     <MC> ::= modify catalog function
     <MF> ::= modify file function
     <PC> ::= purge catalog function
     <PF> ::= purge file function
     <RC> ::= release catalog function
     <RF> ::= release file function
     <LC> ::= list catalog function
     <LS> ::= list specific function
```

The initial communication from ACCESS, following subsystem selection, is a request for a choice of function; i.e., FUNCTION?.

The functions that may be requested and the effect produced by each function are as follows (function may be spelled out or abbreviated as indicated by the underlining):

ACCESS FILE	- Brings a file into the Available File Table.
CREATE CATALOG	- Creates a subcatalog.
CREATE FILE	- Defines file space and attributes for a given file name.
DEACCESS FILE	- Takes a file out of the Available File Table.
LIST CATALOG	- Lists the names of the catalogs and files which emanate from this catalog.
LIST SPECIFIC	- Lists in detail the description of the catalog or file specified.
MODIFY CATALOG	- Modifies the name, password, and/or permissions associated with a given catalog.
MODIFY FILE	- Modifies the name, maximum size, password, and/or permissions associated with a given file.
PURGE CATALOG	- Deletes a catalog from the system along with any subcatalogs and files which are subordinate to it. All released file space is overwritten.
PURGE FILE	- Deletes a file from the system, overwriting the released file space.
RELEASE CATALOG	- Deletes a catalog from the system, along with any subcatalogs and files which are subordinate to it. Any released file space is not overwritten.
RELEASE FILE	- Deletes a file from the system, but without overwriting the released file space.

Each function requires a series of responses from the user. The short form allows the user to supply the responses on a single line preceded by the function. Typical prompts are (as supplied by ACCE):

```
CATALOG STRUCTURE TO WORKING LEVEL?
NEW CATALOG NAME?
PASSWORD?
GENERAL PERMISSIONS?
SPECIFIC PERMISSIONS?
ACCESS FILE?
FILE NAME, SIZE (IN LLINKS), MAX SIZE, MODE?
```

The responses can be seen in the following short form response:

FUNCTION? CF,/CAT1$ABC/CAT2$AOK/FIL1,B/4,~~4~~12~~8~~,R,AF

ACCESS
ACCE

The function is CREATE FILE; the catalog is CAT1 with password ABC, the subcatalogs CAT2 with password AOK; the file name is FIL1 a file with an initial size of 4 blocks (B) and a maximum of 12 blocks; it has read (R) permissions as well as access for the user (AF).

DEVICE/name or type/	request a specific device name or type specified as follows: DSS170 DSS180 DSS181 DSS190 DSS191 DSS270 MS0310 MS0400 MS0450 MS0500 MS0501
AF OPEN	access file after creating it
CLEAR	zero (erase) file space after creating and accessing it

Access type and mode are defined under each applicable function description. Options may appear, comma-separated, in any order. The keywords BLOCKS and LINKS may be abbreviated to the first letter, as may the access-type and mode options. Options unique to the Modify Catalog and Modify File functions are described along with those functions.

All replies may be extended to two or more typing lines by terminating a line with a word delimiter (slant, comma, or dollar sign plus carriage return), at a convenient point, implying that the input is not complete but is to be carried over to the next line or lines.

QUESTIONS AND RESPONSES

Sets of questions associated with each function follow, along with the general form of the response to each question. The minimum required user response is underlined for illustrative purposes. Each set is followed by examples.

ACCESS
ACCE

CREATE CATALOG

FUNCTION? CC

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$pasword/.../cat-name$pasword (qualified catalog names)

NEW CATALOG NAME? cat-name

PASSWORD?
[redacted]

GENERAL PERMISSIONS? access-type,...,access-type

SPECIFIC PERMISSIONS? access-type,...,access-type/user-ID/user-ID/...

The access-types follow; all may be spelled out, or abbreviated as underlined; except for EXCLUDE and LOCK, which must be spelled out:

Permission	Acceptable Abbreviation	Attaches Permission(s)
READ	R	R,E
WRITE	W	R,W,A,E
APPEND	A	A
EXECUTE	E	E
PURGE	P	R,W,A,E,P,REC
MODIFY	M	R,W,A,E,P,M,LOCK,C,REC
LOCK	(none)	LOCK
CREATE	C	C
RECOVERY	REC	R,W,A,E,REC
EXCLUDE	(none)	EXCLUDE (specific permission only)

If no response to the question SPECIFIC PERMISSION? is given, (i.e., only a carriage return), the catalog is created and the question NEW CATALOG NAME? is reissued.

Example replies (user responses are underlined):

FUNCTION? CC

CATALOG STRUCTURE TO WORKING LEVEL?

JDOE/CAT1$ABC

ACCESS
ACCE

This response states that there is a subcatalog named CAT1 that is concatenated directly to the user's master catalog identified by the user-ID JDOE, and that it is desired to create a new catalog from this level. The password ABC was attached to catalog CAT1 when it was created.

NEW CATALOG NAME? CAT2

This response indicates the name of the catalog, CAT2, created at this point.

PASSWORD?
XX@KWEMWEKSKWM

The response, AOK, is entered on the strikeover mask, indicating that this is the desired password. (A carriage return only response would indicate no password.)

GENERAL PERMISSIONS?

The carriage return only response here indicates that general permission is not granted at this level. A response of READ would indicate that any unspecified user has permission to read and execute (if meaningful) any file that emanates from this catalog.

SPECIFIC PERMISSION? READ/BJONES/ASMITH

SPECIFIC PERMISSION? WRITE/WHITE

This combination of responses states that the users who have logged onto the system under the names BJONES and ASMITH can pass through this level with read or execute permission for any files below, and that the user WHITE can pass through with read, write, execute, and append permissions.

SPECIFIC PERMISSION?

The carriage return alone means that no further specific permissions are to be given; the catalog is now created and the question

NEW CATALOG NAME?

is reissued, allowing the user to create another catalog at the same level (i.e., also emanating from CAT1).

Alternative forms of the response to CATALOG STRUCTURE TO WORKING LEVEL? are as follows:

/CAT1$ABC

ACCESS
ACCE

Assuming the user to be JDOE, this response is equivalent to the one given above, JDOE/CAT1$ABC. The initial slant indicates the user's own master catalog.

A response of / indicates that the user desires to create a structure directly subordinate to the User Master Catalog (UMC). This response is equivalent to specification of only the user-ID alone.

Example of short form reply:

FUNCTION? CC,/CAT1$ABC/CAT2,PASSWORD/AOK/,READ/BJONES,
MORE? ASMITH/,WRITE/ALLONG/

CREATE FILE

FUNCTION? CF

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (qualified catalog name)

FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE?

file name, initial size (llinks), maximum size (llinks), mode (R or S)

PASSWORD?
XXGKWBMWEKXKWM

GENERAL PERMISSIONS? access-type,...,access-type

The access types are the same as those for Create Catalog.

SPECIFIC PERMISSION?

access-type,...,access-type/user-ID.../user-ID

Random File Specification: If required, a file can be created with a random-access-treatment indication, by responding to the FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE? question as follows:

file name, initial size, max. size, R

If random (R) is specified, a further question will be asked:

LOGICAL RECORD SIZE? record size in words

Random I/O files for Time Sharing FORTRAN may have a logical record size attribute; if use of random files does not require this attribute, a response with a carriage return only is required.

ACCESS FILE? <u>YES</u>, <u>Y</u>, <u>CLEAR</u>, or <u>C</u>

This option allows the user to access (open) a file at the time it is created. If CLEAR or C is specified, the file space will be zeroed.

Example replies (user responses are underlined):

FUNCTION? <u>CF</u>

CATALOG STRUCTURE TO WORKING LEVEL?

<u>/CAT1$ABC/CAT2$AOK</u>

This response defines user-ID/CAT1/CAT2 as the catalog from which the file is to emanate. The initial slant indicates that the succeeding qualifier string is concatenated to the user's own master catalog.

FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE? <u>FIL1,4,12</u>

This response asks for a file space of four llinks initially, with a maximum eventual size limit of 12 llinks, named FIL1. Since mode is not specified, the file will be created for sequential (linked) usage.

PASSWORD?
X$X#K$$XXXXXXX (null response given)

No password is assigned to this individual file.

GENERAL PERMISSIONS? <u>READ</u>

SPECIFIC PERMISSION?

None is granted at this level, but those granted at the level of CAT2 (CREATE CATALOG in the previous example) apply to this file.

ACCESS FILE? <u>YES</u>

This option allows the user to access (open) a file at the time it is created.

FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE?

This permits creation of other files at the same level.

ACCESS
ACCE

Example of short form reply:

FUNCTION? CF,/CAT1$ABC/CAT2$AOK/FIL1,B/4,12/,R,AF

NOTE: File mode by default is linked (sequential); i.e., MODE/SEQ/.

ACCESS FILE

FUNCTION? AF

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (qualified catalog name)

FILE NAME? filename$password"altname"

PERMISSIONS DESIRED?

access-type,...,access-type

The following table summarizes the legal permission combinations:

Type of Allocation Word	Abbrev.	Allowable Operations on File Content	File Conditions Required	Permissions Required
READ	R	read	no writers, not abort locked	READ
WRITE READ,WRITE	W R,W	read and write	no other writers, not abort locked	WRITE
APPEND	A	append	no writers, not abort locked	APPEND
EXECUTE	E	execute	no writers, not abort locked	EXECUTE
READ,APPEND	R,A	read and append	no writers, not abort locked	READ and APPEND
RECOVERY	REC	read and write	no other writers	RECOVERY
QUERY	Q	read	none	READ
READ, CHANGING	R,C	read	not abort locked	READ
TEST	T	read and write to scratch file	no writers, not abort locked	READ
TEST, CHANGING	T,C	read and write to scratch file	not abort locked	READ
WRITE,C READ,WRITE,C	W,C R,W,C	read and write	not abort locked	WRITE

Random File Specification: A file can be accessed for random treatment, whether created as random or linked, by responding to the FILE NAME? question with:

filename$password,R

or

filename$password"altname",R

If the file was created as linked, the random treatment indication is temporary; i.e., for the current access only. If the file was created as random, the ,R specification is superfluous.

Example replies (user responses are underlined):

FUNCTION? AF

CATALOG STRUCTURE TO WORKING LEVEL?

JDOE/CAT1$ABC/CAT2$AOK

The user in this case is not the creator of the file to be accessed, so the user must define the other user's master catalog (e.g., JDOE) from which the file emanates, along with any required subcatalogs and passwords.

FILE NAME? FIL1

If a password were required, it could be concatenated to the name with a dollar sign; i.e., FIL1$ABC. Otherwise, it will be requested.

PERMISSIONS DESIRED? READ

General Read permission was granted for this file. (Several specific Read permissions were also granted at the level immediately above CAT2.) Termination of this response with only a carriage return causes the file to be accessed and the request

FILE NAME?

to be reissued.

Example of short form reply:

FUNCTION? AF,JDOE/CAT1$ABC/CAT2$AOK/FIL1,R

DEACCESS FILES

FUNCTION? DF

FILE NAME? filename (or CLEARFILES, PERMFILES, STARFILES, or TEMPFILES)

The response for this function is the name of the file to be deaccessed. The name supplied is always the name under which the file was accessed, whether this was the actual name or a temporary alternate name. If CLEARFILES is used, all of the user's available files (except *SRC and SY**) are deaccessed including temporary files. PERMFILES or TEMPFILES may be used to remove all permanent or temporary files (except *SRC and SY**) from the AFT, respectively. STARFILES removes all files (except *SRC and SY**) from the AFT that contain an asterisk in the name. Note that the input collector file (SY**) will never be deaccessed.

Example of short form reply to deaccess a file that was created in an earlier example:

FUNCTION? DF,FIL1

PURGE CATALOG

FUNCTION? PC

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (qualified catalog name)

CATALOG TO BE PURGED? cat-name$password

The dollar sign is used only when the password is concatenated directly to a file or catalog name.

Example replies (user responses are underlined):

FUNCTION? PC

CATALOG STRUCTURE TO WORKING LEVEL?

/CAT1$ABC

This response defines the subcatalog CAT1 concatenated to the user's own master catalog.

CATALOG TO BE PURGED? CAT2$AOK

(The catalog and all catalogs and files subordinate to it are now purged.)

CATALOG TO BE PURGED?

is reissued.

Example of short form reply to purge a catalog that was created in an earlier example:

FUNCTION? PC,/CAT1$ABC/CAT2$AOK

PURGE FILE

FUNCTION? PF

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (qualified catalog name)

FILE TO BE PURGED? file name$password

Password request will be issued if incorrectly given or omitted.

Example replies (user responses are underlined):

FUNCTION? PF

CATALOG STRUCTURE TO WORKING LEVEL?

JDOE/CAT1$ABC/CAT2$AOK

The user in this case is ALLONG, not the file creator.

FILE TO BE PURGED? FIL1

(The file is now purged.)

The request

FILE TO BE PURGED?

is reissued.

ACCESS
ACCE

Example of short form reply to purge a file that was created in an earlier example:

FUNCTION? PF,JODE/CAT1$ABC/CAT2$AOK/FIL1

RELEASE CATALOG

FUNCTION? RC

The question/response sequence and the short form reply for this function are completely analogous to those for the Purge Catalog function. The Release Catalog function would normally be used in preference to Purge Catalog -- as it is more economical -- unless the user has a very stringent file-security requirement.

RELEASE FILE

FUNCTION? RF

The question/response sequence and the short form reply for this function are completely analogous to those for the Purge File function. The Release File function would normally be used in preference to Purge File -- as it is more economical -- unless the user has a very stringent file-security requirement.

MODIFY CATALOG

FUNCTION? MC

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/...,/cat-name$password (qualified catalog name)

CATALOG TO BE MODIFIED? cat-name and password (if needed)

NEW NAME? new cat-name

NEW PASSWORD? { new password | DELETE }
MNSEZDEKNBNEDE

GENERAL PERMISSIONS? { access-type,...,access-type | DELETE }

SPECIFIC PERMISSION? { access-type,...,access-type/ user-ID.../user-ID | DELETE/user-ID/.../user-ID }

ACCESS
ACCE

Example replies (user responses are underlined):

FUNCTION? MC

CATALOG STRUCTURE TO WORKING LEVEL? /CAT1$ABC

CATALOG TO BE MODIFIED? CAT2$AOK

NEW NAME?

A carriage return only response means that the catalog name is to remain unchanged.

NEW PASSWORD?
XXXXXXXXXXXXXX (response "XYZ" given)

The original password AOK is replaced by XYZ.

GENERAL PERMISSIONS? READ

As originally created, general permissions were not assigned at this level. This response replaces this null set with READ and EXECUTE permission.

SPECIFIC PERMISSION? W/BJONES

This response replaces the original specific READ permission for BJONES with READ, WRITE, EXECUTE and APPEND permission.

SPECIFIC PERMISSION? DELETE/ASMITH

This response cancels any permissions for ASMITH that previously existed.

SPECIFIC PERMISSION? P,LOCK/WHITE

This response replaces the original set of permissions for WHITE with PURGE and LOCK.

SPECIFIC PERMISSION?

The carriage return implies that no further modifications are to be made; the changes are now processed and the question

CATALOG TO BE MODIFIED?

is reissued.

ACCESS
ACCE

Special Short Form Option Formats

To rename a catalog:

NEWNAME/catalog/ or N/catalog/

To exclude, by user-ID, from any general permissions:

EXCLUDE/user-ID,..., user-ID/

To delete specific permissions, by user-ID:

DELETE/user-ID,...,user-ID/

To delete all general permissions:

DELETE/GEN'L/(or simply DELETE)

NOTE: EXCLUDE and DELETE may not be abbreviated.

Example of short form reply:

FUNCTION? <u>MC,/CAT1$ABC/CAT2$AOK,PASSWORD/XYZ/,W/BJONES/,DELETE/ASMITH/</u>

MORE? <u>P/WHITE/,LOCK/WHITE</u>

MODIFY FILE

FUNCTION? <u>MF</u>

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (qualified catalog name)

FILE TO BE MODIFIED? filename and password (if needed)

NEW NAME? new filename

NEW MAX SIZE? new maximum size (in llinks)

NEW PASSWORD? $\left\{ \begin{array}{l} \text{new password} \\ \text{DELETE} \end{array} \right\}$
XXXXXXXXXXXXXX

ACCESS
ACCE

$$\text{GENERAL PERMISSIONS?}\ \left\{\begin{array}{l}\text{access-type,...,access-type}\\ \text{DELETE}\end{array}\right\}$$

$$\text{SPECIFIC PERMISSION?}\ \left\{\begin{array}{l}\text{access-type/user-ID/.../user-ID}\\ \text{DELETE/user-ID/.../user-ID}\end{array}\right\}$$

Example replies (user responses are underlined):

FUNCTION? MF

CATALOG STRUCTURE TO WORKING LEVEL?

/CAT1$ABC/CAT2$XYZ

FILE TO BE MODIFIED? FIL1

NEW NAME? MASTER1

NEW MAX SIZE? 20

This response increases the maximum file size to 20 llinks (originally 12).

NEW PASSWORD?
XMEKEKGKIXKKMX (response "DEPT37" given)

This response attaches the password DEPT37 (which would be in the strikeover area) to this file (none originally assigned).

GENERAL PERMISSION? DELETE

The original general READ permission is deleted.

SPECIFIC PERMISSION? P/BJONES

PURGE permission for user BJONES is added at this level. This permission applies to this file only.

Special short form option formats:

To rename a file:

NEWNAME/filename/or N/filename/

To exclude, by user-ID, from any general permissions:

EXCLUDE/user-ID,...,user-ID/

ACCESS
ACCE

To delete, by user-ID, specific permissions:

DELETE/user-ID,...,user-ID/

To delete all general permissions:

DELETE/GEN'L/

or

DELETE

NOTE: EXCLUDE and DELETE may not be abbreviated.

To change the mode of a file:

MODE/mode/

Example of short form reply:

FUNCTION? MF,/CAT1$ABC/CAT2$XYZ/FIL1,N/

MORE? MASTER1/,B/20/,PASS/DEPT37/,DELETE,P/BJONES/

LIST CATALOG

FUNCTION? LC or LIST CATALOG

CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?

user-ID/cat-name/,...,/cat-name,n,x(mm-dd-yy),S,A,R,FIRST/name/

Passwords need not be given in the catalog structure unless the catalog to be listed was created by another user. A user may list only catalogs created by him, or the Library catalog (#LIB) or, the command library catalog (#CMD), or catalogs belonging to other users for which the user has modify permission.

The List Catalog provides selective listing of catalog and file names by the use of the optional parameters n,x(mm-dd-yy),S,A,R, FIRST/name/ where:

x, specifies whether date is date created (C), date of last access (A), or last date the file contents were changed (L)

mm-dd-yy, starting date for C, A, or L option

n, number of files to be listed

S, sort the names

A, abbreviated list (eight per line)

R, reverse the order of printing

FIRST/name/ starts the catalog listing at the specified catalog or file name.

Any option may be omitted and the order in which they are given is immaterial.

Examples (user responses are underscored):

FUNCTION? <u>LC</u>

CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?

<u>/CAT1</u>

Requests a list of the catalog and files emanating from CAT1.

<u>/CAT1,C(01-01-79)</u>

Requests a list of all catalog and file names created under CAT1 since January 1, 1979.

<u>/CAT1,A(07-01-79),R</u>

Requests a list of all catalog and file names that emanate from CAT1 and were accessed since July 1, 1979 and in reverse order (most recent to oldest).

<u>/CAT1,L(06-01-79),10</u>

Requests a list of the first ten catalog and file names that emanate from CAT1 and whose contents were changed since June 1, 1979.

/CAT1,R,10

Requests a list of the ten most recently created catalog and file names emanating from CAT1.

/CAT1,10

Requests a list of the ten oldest catalog and file names emanating from CAT1.

/CAT1,FIRST/FILX/

Requests a list of catalog and file names emanating from CAT1, starting at FILX.

LIST SPECIFIC

FUNCTION? LS

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name/,...,/cat-name

CATALOG OR FILE TO BE LISTED?

CATALOG OR FILE NAME

Example replies (user responses are underlined):

FUNCTION? LS

CATALOG STRUCTURE TO WORKING LEVEL?

/CAT1

CATALOG OR FILE TO BE LISTED? FIL1

Passwords need not be given in the catalog structure unless the specified file or catalog was created by another user.

The description of FIL1 would now be listed.

ACCESS
ACCE

The system will provide the following information (but not the password) about the catalog or file:

FILE NAME-
ORIGINATOR-
DATE CREATED-
DATE CHANGED-(month/day/year plus time of day in parentheses)
LAST DATE ACCESSED-
NUMBER OF ACCESSES-
MAX FILE SIZE-
CURRENT FILE SIZE-
FILE TYPE-RANDOM,LINKED or I-D-S
DEVICE-
GENERAL PERMISSIONS-
SPECIFIC PERMISSIONS-

In addition to a list of the user's own catalogs or files, a user may obtain a specific list of the library (#LIB), the command library (#CMD), or the catalogs or files belonging to other users for which the user was the creator or has modify permission.

EXAMPLES OF LINE DELIMITER USE

The line delimiters can be used in several ways either to shorten the question/response sequence or terminate a function at any given point.

Examples of the effect of different response terminations are as follows:

FUNCTION? CC

CATALOG STRUCTURE TO WORKING LEVEL?

The carriage return alone implies a master catalog.

NEW CATALOG NAME? 001*

Passwords or permissions are not wanted for this catalog and no further questions are wanted. Return is to NEW CATALOG NAME? level.

NEW CATALOG NAME? 002

PASSWORD?
XXXXXXXXXXXXXX (PASS2**)

No permissions are to be assigned to this catalog, and creation of catalogs at this position is finished. Return is to function level.

FUNCTION? CF

CATALOG STRUCTURE TO WORKING LEVEL?

/002$PASS2

FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE? 02.1,1,3

PASSWORD?
MMMMMMMMMMMM (null response given)

GENERAL PERMISSIONS? READ

SPECIFIC PERMISSION? W/RJJONES**

Creation of files at this level has been completed.

FUNCTION? carriage return (or DONE)

Finished with ACCESS.

*

Return to the subsystem selection level.

SPECIAL FEATURES

Files created by means of the Create File function are not necessarily contiguous; i.e., successive llinks of a file are not necessarily in physical sequence on the storage device. Furthermore, both the Create File and Access File functions assume that the file will be treated as a linked file. For the standard subsystems provided with the Time Sharing System, these file characteristics are suitable because linked files are most commonly used.

If, however, in the use of a given subsystem, it would be advantageous to have contiguous files, this characteristic can be specified in response to FILE NAME,SIZE(IN LLINKS),MAX SIZE,MODE?. The form of this response is:

filename,initial size C

The parameter C indicates, in Create File only, that a contiguous file is desired. No maximum size may be specified.

Similarly, if random treatment of files is required in a given user-written subsystem, a file can either be created as a random file or accessed as a random file. If created as such, it is always treated by the GCOS I/O Supervisor as a random file. If it is created as a linked file, it can be accessed as a random file, but in that case, the random treatment indication is temporary; i.e., it applies to that access only.

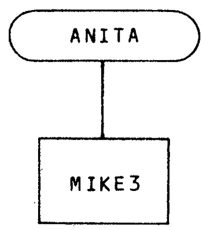
ACCESS
ACCE

Example

Type all underline responses and note the general comments. To test a catalog structure it will be helpful to draw block diagrams as we go. Since our user-ID is ANITA we will start at that level.

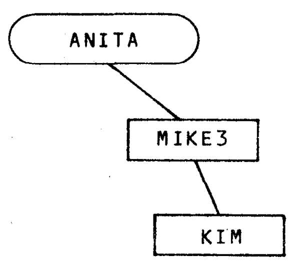
Enter subsystem access and create a subcatalog.
*ACCESS
FUNCTION? CC CREATE A CATALOG.
CATALOG STRUCTURE TO WORKING LEVEL?
ANITA Identify the USER ID
NEW CATALOG? MIKE3 The new name is MIKE3
PASSWORD? KARLA3 MIKE3's Password is KARLA3
GENERAL PERMISSIONS? R,W,E,A The permissions are READ, WRITE, EXECUTE, APPEND
SPECIFIC PERMISSIONS? (carriage return) None specified
NEW CATALOG? (carriage return) stop adding catalogs at this level

RESULT --



FUNCTION? CC
CATALOG STRUCTURE TO WORKING LEVEL?
ANITA/MIKE3$KARLA3 Add a catalog to the catalog MIKE3 - lower level
NEW CATALOG? KIM
PASSWORD? TIM
GENERAL PERMISSIONS? R,W,E,A
SPECIFIC PERMISSIONS CR
NEW CATALOG? CR

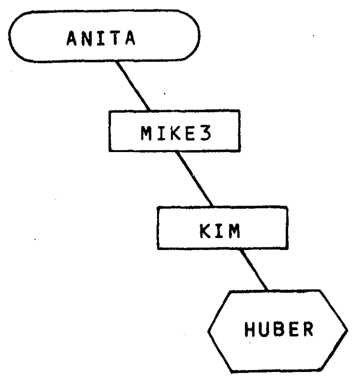
RESULT --



ACCESS
ACCE

```
FUNCTION?  CF                          Create a file
CATALOG STRUCTURE TO WORKING LEVEL?
/MIKE3$KARLA3/KIM$TIM                  Place this file under the catalog KIM
FILE NAME, SIZE (IN FLCKS),MAX SIZE?
HUBER,12,36                            The name is HUBER and it is a min. of 1 link
                                       and a max. of 3
PASSWORD?  JOE
GENERAL PERMISSIONS?  R,W,*            Permissions are read and write, the asterisk
                                       indicates you are through with your responses -
                                       process this information and return to the first
                                       question.
FILE NAME, SIZE (IN BLKS) MAX. SIZE?  CR
```

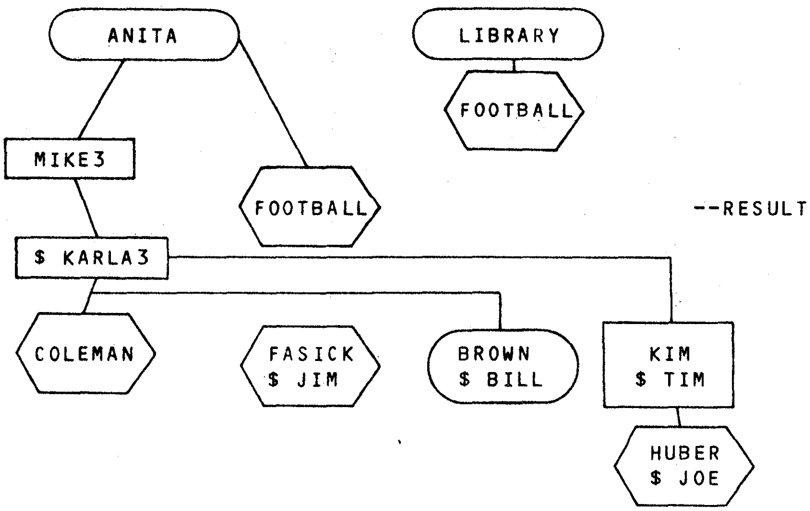
RESULT--




```
FUNCTION?  CF
CATALOG STRUCTURE TO WORKING LEVEL?
/MIKE3$KARLA3                          Files will be added to catalog MIKE3
FILE NAME, SIZE (IN BLKS) MAX. SIZE?
COLEMAN,1242                           File name COLEMAN, min. 1 link, max. 2 links
PASSWORD?  CR                          Do not assign a password
GENERAL PERMISSIONS?  R,W,*            Return to original question
FILE NAME, SIZE (IN BLKS) MAX. SIZE?
FASICK,12,12                           File is FASICK with 1 link min. and max.
PASSWORD?  JIM*                        Password is JIM - return to original question
FILE NAME, SIZE (IN BLKS) MAX. SIZE?
BROWN,12,24                            File is BROWN 1 to 2 links
PASSWORD?  BILL**                      Password is BILL - double asterisks indicate a
                                       completion of this series of questions, return
                                       to the function level.


FUNCTION?  AF                          This will access a file
CATALOG STRUCTURE TO WORKING LEVEL?
LIBRARY                                Access a library file
FILE NAME $ PASSWORD?  FOOTBALL
PERMISSIONS DESIRED?  R*               Read permission desire, then return to original
                                       question.
FILE NAME $ PASSWORD?  DONE            The word DONE given at any level will return
                                       you to system.
*AFT                                   Request a listing of open files to see if
                                       FOOTBALL is added.
```

At this point lets look at the complete catalog-file outline with passwords.




```
*ACCE                               Re-enter access subsystem
FUNCTION?  DF                       Deaccess a file
FILE NAME?  FOOTBALL**
FUNCTION?  DONE
*STATUS                             If the file FOOTBALL was deaccessed there will
                                    not be any open files.


*ACCE
FUNCTION?  LC                       List a catalog
CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?
/MIKE3                              List from the catalog level MIKE3
FUNCTION?  LC
CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?
/MIKE3$KARLA3/KIM                   List from catalog level KIM down.
Now that this catalog/file mountain is built, let's tear it down:


FUNCTION?  PC                       Purge a catalog
CATALOG STRUCTURE TO WORKING LEVEL?
/MIKE3$KARLA3                       Catalog to be purged is below the level (MIKE3)
CAT. TO BE PURGED?  KIM
PASSWORD?  TIM
NOTE:  THIS WIPES OUT CATALOG KIM AND ALL THE FILES UNDER KIM (HUBER $ JOE)


CATALOG STRUCTURE TO WORKING LEVEL?  CR
FUNCTION?  PF                       PURGE A FILE
CATALOG STRUCTURE TO WORKING LEVEL?
ANITA/MIKE3$KARLA3                  The file is assigned to MIKE3 catalog
FILE TO BE PURGED?   COLEMAN
PASSWORD?  CR
FILE TO BE PURGED?  CR
```

Purpose

The ADMN command gives the System software administrator, whose user-ID is ..SYS ADMIN, the ab lity to control the FMS structure necessary to support Selectable Unit (SU) packaging.

Format

ADMN

Description

The commands of the ADMN subsystem are:

CREATE - Creates user-ID required to hold a Selectable Unit (SU).

RELEASE - Deletes a SU version by releasing the space it occupies back to FMS.

INSTALL - Installs a previously loaded SU version as the production version.

DESTROY - Deletes an entire SU (opposite of CREATE)

AUDIT - Produces a report of all the installed software.

DONE - Used to exit the System Administrators interface.

Purpose

The AFT command displays the contents of the Available File Table (AFT).

Format

AFT

Discussion

The AFT command provides a convenient display of all files presently open in the user's available file table. The display lists the file names in a condensed manner with multiple entries per line. The AFT command allows the user to quickly determine the presence or absence of a file from the available file table. Both temporary and permanent files that are present are listed.

Examples

```
*AFT
   SY**     *SCR     *CFP     TEMP1
*GET TEMP2
*AFT
   SY**     *SCR     *CFP     TEMP1     TEMP2
```

Purpose

The ALGOL command invokes the subsystem interface to the ALGOL compiler.

Format

ALGO[L]

Discussion

The ALGOL time sharing system provides the capability for compiling, loading, and executing ALGOL programs. Refer to ALGOL manual (DD27) for further details.

Purpose

The APB command allows a user to retrieve the last All Points Bulletin issued by either the host console operator or the MASTER user.

Format

```
APB
```

Discussion

If no message exists, an immediate return to the previous level of processing is taken. If a message exists in the time sharing executive communication region buffer area, a DRL T.CMOV is executed to obtain the message for subsequent display formatting at the terminal.

Example

```
1)    *APB

2)    *APB
        **12.015** TEST  MESSAGE
```

Purpose

The APRINT command invokes the CONVERT subsystem to generate an ASCII printer report of a TSS file. Typically the output file from RUNOFF is a candidate for APRINT.

Format

```
APRI[NT] [INFILE] [:OPTIONS]

   [INFILE]::=<filedesc>
   [OPTIONS]
```

Description

Media Code Options

The output record format options specify the physical format of the output record. The default option for the CONVERT command is "ASCII". A list of the options and their meanings is as follows:

BCD	- variable-length BCD - media code 0
COMDK	- BCD compressed deck card image (COMDK) - media code 1
CARD	- BCD 14-word card image - media code 2
PRINT	- BCD variable-length print line image - media code 3
OLDASC	- obsolete TSS ASCII - media code 5
ASCII	- standard system format ASCII - media code 6
APRINT	- ASCII print line image - media code 7
ACARD	- ASCII card image - media code 10
SAME	- a record output media code is the same as its input media code

Line Number Options

Line numbers can exist with COMDK, CARD, ACARD, OLDASC, and ASCII records. All BCD, PRINT, and APRINT records cannot possess line numbers. The line number for an ASCII or OLDASC record consists of 1 to 8 numeric characters. These numeric characters must be among the first eight characters in a line. A line number is defined to include any leading blanks. A line number is terminated by a nonnumeric character, including blank. If the "#" character terminates a line number and if it is one of the first eight characters of a line, it is considered to be a delimiter. It is treated as neither part of the line number nor part of the text. The line number for COMDK, CARD, and ACARD records is defined to be all the trailing digits in columns 73-80. This field may begin with nonnumerics; these also are considered neither part of the line number nor part of the text.

The line number options may specify:

1. Whether line numbers are to appear in the output text.

2. The actual line number values.

The default line number option is "ASIS". A description of each of the options follows:

ASIS — Line numbers are assumed not to be present in the input file. Text, including leading/trailing numeric characters and "#"'s are left as is.

STRIP — Strip line numbers from the input text before reformatting and writing the output text. Input COMDK, CARD, and ACARD records are truncated at column 72. Line numbers on ASCII and OLDASC records, when present, are discarded and the first character following the line number is treated as the first character of the line.

MOVE — Move line numbers. The input records have the line numbers detached from the text string, either from the front (ASCII or OLDASC) from columns 73-80 (COMDK, CARD, or ACARD). The output records have the line numbers reattached to the text string, either at the front (ASCII or OLDASC) or in columns 73-80 (COMDK, CARD, or ACARD). If the output records are BCD, PRINT, or APRINT, the line numbers are not reattached and the M option acts similar to the S option.

I(i,j) — Insert line numbers beginning with line number i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. The input file is assumed not to be line-numbered. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the I option is ignored.

R(i,j) Resequence line numbers. Strip any existing line numbers from the input text and insert new line numbers in the output text, beginning with i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the R option behaves as the S option.

N(ch) Implies the M option and specifies that the normal tab character (the colon) and tab settings (8, 16, 32, 73) have been employed in building the input file(s). The (ch) argument may be used to define a character which replaces the colon as the tab character.

LABEL (abcde(i-j)fghij(i-j)---) If the output records are COMDK, CARD, or ACARD, then a label is placed left-justified in columns 73-77. The label is specified as 1 to 5 nonblank characters. The fields "abcde" and "fghij" represent the labels. The label is placed on only those lines with line numbers between i and j inclusive. Up to 10 distinct labels may be given. If more than one label is given though, the (i-j) specifications may not overlap.

The LABEL option is meaningful only if line numbers are attached to output records. Therefore, the label option is completely ignored unless it is accompanied by either the insert, resequence, or move option.

For the I and R options, output line numbers for ASCII and OLDASC records will have at least the number of digits specified for i in I(i,j) or R(i,j). Thus R(0010,10) will result in line numbers 0010, 0020, 0030,---.

Input records are assumed to have line numbers when the STRIP, MOVE, and RESEQUENCE options are specified. Otherwise, line numbers are assumed to be absent and leading numerics in ASCII format are treated as real text. When line numbers are assumed present, tabbing and columnizing are performed relative to the start of the real text.

The user must be careful not to alter the line number values of a BASIC file.

Character Manipulation Options

A description of each of the character manipulation options follows.

TAB(ch,i,j---;ch,i,j---;----) Expand tab characters into blanks. Use "ch" as a tab character with settings i,j,k,etc. Usually, any occurrence of the tab character in the input file(s) results in the replacement of the tab character with a string of blanks up to the next tab setting. However, if a tab character is encountered beyond the last tab setting specified for that tab character, it is treated as a normal nontab character.

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the tab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

UNTAB(ch,i,j---;ch,i,j---;----) Insert tab characters, replacing blanks. Use "ch" as a tab character with settings i, j, k, etc. Any occurrence of a string of blanks terminating on an "untab" tab stop is replaced by the character "ch".

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the untab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

LOWER — Convert all alphabetic characters to lowercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

UPPER — Convert all alphabetic characters to uppercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

BEGIN(ch) — Begin a new line (record) immediately after the character "ch". The character "ch" is treated as a delimiter and not part of the text. It is not placed in the output text. When the "ch" character is located at the beginning or end of a line, it is simply deleted. Strings of the "ch" character are treated as a single "ch" character.

COLUMNS(i-j) — Delete all of the characters in a line except those which are located within columns i through j inclusively. The options BEGIN and TAB are both completed before COLUMNS takes effect. If a record does not extend through column j prior to the COLUMNS option execution, it is blank filled to column j. Thus, when the COLUMNS options is in effect, the length of all generated output records is j-i+1 characters.

SQUEEZE — Replace any string of two or more blanks by a single blank. The options BEGIN, TAB, COLUMNS, and UNTAB are all performed before SQUEEZE is executed.

TRAIL — Delete all trailing blanks on a line. The TRAIL option is performed immediately after the SQUEEZE option.

A number of options affect the length of an output text line. It is important that the user understand the order in which these options are performed. The order (from first to last) in which the options are executed is:

BEGIN
TAB
COLUMNS
UNTAB
SQUEEZE
TRAIL

Miscellaneous Options

VERIFY — If the VERIFY option is in effect when CONVERT completes the processing of an input file, then CONVERT gives a brief summary of the number of records obtained from the file. This summary gives, for each media code, the number of records which had that media code.

IGNORE — Ignore all embedded $$ control lines. Treat them as text.

DISCARD — Discard all nontext records. Nontext records are those records whose media code is not one recognized and interpreted by CONVERT. The JRN, JPRINT, JPUNCH, APRINT, and DISPLAY commands require that nontext records be discarded. The CONVERT command normally does not require that nontext records be discarded. When nontext records are encountered during the execution of the CONVERT command, they are written to the output file, but no reformatting or media conversion is performed.

TIME — When the TIME option is invoked, the date and time of day are printed at the user's terminal.

DEFAULT — The DEFAULT option is used to nullify all options which the user has specified either on the command line or embedded $$ control lines. The default option has no affect on any of the "specialized" options. Because of the nature of the DEFAULT option, it is meaningless for it to be located in the options field of the command line. Therefore, if the DEFAULT option is encountered in the options field, an error message is issued. The same reasoning applies to the placement of the DEFAULT option anywhere other than the beginning of a $$ control line.

File Processing Options

SELECT (file) — The SELECT option is analogous to the $ SELECTA card. The select option allows an input file to specify other input files. Upon encountering the SELECT option, the selected file is obtained and is used in place of the $$ control line. Nesting of selects is permitted up to 17 levels. The SELECT option is meaningful and valid only on a $$ control line. Only one SELECT option may be specified on a $$ control line.

INCLUDE — If the INCLUDE option is in effect, CONVERT, upon encountering the SELECT option, uses the selected file as an input file.

EXCLUDE — If the EXCLUDE option is in effect, CONVERT ignores the SELECT option.

The purpose of the INCLUDE and EXCLUDE options is to allow the user to control the performance of the select options while not forcing him to disregard:

1. Other options on the same $$ control line.

2. All $$ control lines.

The INCLUDE option is the default option for the JRN command. The EXCLUDE option is the default option for the JPRINT, JPUNCH, APRINT, DISPLAY, and CONVERT commands.

Specialized Options

The "specialized" options are a class of options completely distinct and separate from all preceding options. The "specialized" options are unlike other options in that they take effect only when all input files have been read, converted, and closed; i.e., after the output file has been completely generated. All other options, of course, are meant to be used when the output file is in the process of being generated.

ROUT(xx) — The ROUT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. This option causes the implied files generated by the program execution to be directed to the specified two-character remote station. Only one ROUT entry is permitted.

WAIT — The WAIT option is applicable to the JRN, JPRINT, APRINT,and JPUNCH commands. This option causes the user to wait until the completion of the spawned job in the batch environment. The wait period may be broken out of by hitting the break key. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

COPY(nn) — The COPY option is applicable only to the JPRINT, APRINT, and JPUNCH commands. This option causes the generation of nn multiple copies of the listing or punched deck. The maximum number of copies that can be produced from a single JPRINT/JPUNCH job is 13.

IDENT(info) — The IDENT option is applicable to the JPRINT, APRINT, and JPUNCH commands. This option allows the user to minimize the subsystem/user interface involved in the use of the JPRINT/JPUNCH commands. When the IDENT option is present, the normal question/answer sequence of

    $ IDENT? response

is bypassed. The information presented as the IDENT option argument is used instead of the user-response to the question.

MONITOR — The MONITOR option is applicable to the JPRINT, APRINT, JPUNCH, and JRN commands. This option allows the user to monitor or track the status of his spawned job as it is executed in the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

DIRECT The DIRECT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. If the DIRECT option is given on the command line, it overrides any JOUT or ROUT option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the JOUT option to place it on a $$ control line. He can then override it without being required to change his $$ control line.

INDENT(99) The INDENT option applies to the APRINT command and specifies the number of print columns to indent from the left margin.

PAGELENGTH(99) The PAGELENGTH option applies to the APRINT command and specifies the print page size for non-RUNOFF files being processed.

The ROUT, JOUT, and DIRECT options are mutually exclusive. The MONITOR, TALK, WAIT, and DISMISS options are also mutually exclusive. Mutually exclusive options are a group of options for which only one member of the group of options may be in effect. If the user attempts to give two mutually exclusive options in the options field of the command line or on a $$ control line, an error message is given.

Example

```
*LIST

.page 1
.pape 66
.head 1,1
TSS DESIGN SPECIFICATION
.spac 3
.cent 1
EXTERNAL DESIGN SPECIFICATION
.spac (LIST interrupted)
*RUNO
ready
REFORM *,TEMPPRT
ready
NOSTOP
ready




runoff complete
DONE
*APRI TEMPPRT:IDENT(P10ACAJ12,L. MILLER,STATION J J J),INDE(10),COPY(2)
SNUMB 2155V
```

Purpose

To convert an media code-5 ASCII format file to the standard, media code-6 ASCII time sharing format; or, convert a type-6 ASCII format file to type-5 ASCII format; or, convert a ASCII time sharing format file to type-5 ASCII format.

Format

```
ASCA[SC][ <infile>;[otfile>]]

  <infile>     ::= *|<filedesc>
  <otfile>     ::= *|<filedesc>
```

Discussion

In the execution of the ASCASC command, the input file is read and converted to the format required for the output file. The input file's record control word is checked to determine the format of the file. If the record media code is 5, the file is in time sharing ASCII format. If the record media code is 6, the file is in standard system ASCII format. Based on this determination, one of the following translations is performed:

1. If the input file is in time sharing ASCII format (character-oriented file), the characters in the file are read and converted to the word-oriented standard system ASCII format for the output file.

2. If the input file is in standard system ASCII format, the words in the file are read and converted to the time sharing ASCII format for the output file. Up to 72 characters will be converted.

If neither record media code 5 or 6 is found in the record control work, a message is sent to the user informing him that the file specified is not an ASCII file.

Examples

1) ASCASC
   (prompts for the names of the two ASCII files)

2) ASCA ASCII5;NEWASCII
   (assuming that the file ASCII5 contains old type-5 ASCII format data, converts it to the new ASCII time sharing format and stores the result in the file NEWASCII)

3) ASCA ASCII6;ASCII5
(assuming that the file ASCII6 contains old type-6 ASCII format data, converts it to the old type-5 ASCII format and stores the result in the file ASCII5)

4) ASCASC NEWASCII;ASCII5
(assuming that the file NEWASCII contains new ASCII time sharing format data, converts it to the old type-5 ASCII format and stores the result in the file ASCII5)

Purpose

To convert an ASCII time sharing format file to a standard system format BCD file.

Format

```
ASCB[CD][ <ascfile>;[<bcdfile>]]

  <ascfile>     ::= *|<filedesc>
  <bcdfile>     ::= <filedesc>
```

Description

<ascfile> is the name of a file or the current file that contains ASCII time sharing format data. The current file is specified by an asterisk.

<bcdfile> is the name of a file that will contain the converted standard system format BCD data. If this file does not already exist, it is created with general READ permission.

Discussion

The characters in the ASCII file are converted to corresponding BCD characters,. For some ASCII characters, however, there are no equivalent BCD characters. Refer to Appendix D for a description of the character transliterations.

A question/answer sequence is initiated by this command unless the ASCII file contains first-line reformatting information.

Examples

1) ASCBCD
   (prompts for the names of the ASCII and BCD files)

2) ASCBCD DATA.A;DATA.B
   (converts the contents of the file DATA.A to BCD and stores the result in the file DATA.B)

3) ASCB *;DATA.B
   (converts the contents of the current file to BCD and stores the result in the file DATA.B)

4) ASCB DATA.A;
   (prompts for the name of the BCD file)

Purpose

To cause the automatic creation of line numbers at the point the automatic mode is entered or reenter d.

Format

```
AUTO[MATIC][X][ <begin-line>][,<increment>]

  <begin-line>      ::= a decimal number
                     |  =
                     |  *
  <increment>       ::= a decimal number
```

Description

<begin-line> sets the starting line number. If <begin-line> is not specified, line numbering will start at 10 on the first entry to automatic mode. On reentry to automatic mode, line numbering will resume where the previous automatic numbering left off. This default for <begin-line> is used in Examples 2, 4, and 6.

If <begin-line> is specified as a number, line numbering will begin with the value of <begin-line>, which would be 10, 40, 100, and 100 respectively, in Examples 1, 3, 5, and 7.

If <begin-line> is specified as =, on first entry to automatic mode, line numbering will begin at 10. On reentry to automatic mode line numbering will begin at the last line number generated by the previous AUTO command minus the last value of <increment>.

If <begin-line> is specified as *, line numbering will begin at 10 for an empty current file or at the last line number of the file plus the <increment> for a nonempty current file.

<begin-line> is affected only by whether or not the entry is the first entry to the automatic mode and whether or not the current file is empty.

<increment> sets the increment between line numbers. If no value for <increment> is given, on the first entry to automatic mode the value 10 will be used. On reentry, the previous value of <increment> will be used.

Discussion

When the AUTO or AUTOMATIC form of the command is used, each line number will be followed by a blank. This would be the case in Examples 1, 2, 3, 8, and 9. If the letter X is affixed to the end of the AUTOMATIC command, (i.e., AUTOX or AUTOMATICX) then the blank following the line number will be suppressed. This would be the case in Examples 4, 5, 6, 7, and 10. Since no blank character follows the line number generated by AUTOX, any numbers typed immediately following the AUTOX line number become a part of the line number.

When AUTO is used, the generated line number is represented by three digits as long as the line number is less than 1000 (e.g., 010, 020, 100, 999). All line numbers greater than 999 are represented by seven digits with 9999999 as the maximum value for a line number (e.g. 0001000, 0001010, 0030000, 9999999).

When AUTOX is used, the generated line number is represented by four digits as long as the line number is less than 1000 (e.g. 0010, 0020, 0100, 0999). All line numbers greater than 0999 are represented by eight digits with 99999999 as the maximum value for a line number (e.g., 000010000, 00010000, 99999999).

The line numbers created by this command appear in the terminal copy, and are written in the current file, just as though the user had typed them.

No commands are recognized while in the automatic mode. The automatic mode is cancelled by giving a carriage return immediately following the issuance of an asterisk and line number by the system. The user may not use a character-delete or a line-delete character to delete characters in the generated line number nor, if AUTO is used, can the blank after the line number be deleted.

In the current "LOGON" session, on exit from the AUTOMATIC command, the current values specified for <begin-line> and <increment> will not be affected by other commands and will be the defaults in a subsequent AUTOMATIC command.

Examples

```
 1)   AUTO 10,10
 2)   AUTOMATIC      (same as AUTO 10,10)
 3)   AUTO 40,5
 4)   AUTOX
 5)   AUTOX  100
 6)   AUTOX ,20
 7)   AUTOMATICX 100,2
 8)   AUTO *
 9)   AUTO =
10)   AUTOX =,4
```

Purpose

The BASIC command invokes the BASIC subsystem for entering, compiling and running BASIC programs.

Format

BASI[C]

Discussion

The BASIC subsystem can be invoked while under another subsystem by entering the BRN run command. For further explanation and examples of BASIC usage, see Section IX or see the description of the BRN command.

Purpose

To convert a standard system format BCD file to an ASCII time sharing format file.

Format

```
BCDA[SC][ <bcdfile>[;<ascfile>]]

  <bcdfile>     ::= <filedesc>
  <ascfile>     ::= *|<filedesc>
```

Description

<bcdfile> the name of a file that contains standard system format BCD data.

<ascfile> the name of a file or the current file that will contain the converted ASCII time sharing format data. If <ascfile> is an asterisk as in Example 3, the output is put into the current file. If the ASCII file does not already exist, it is created with general READ permission.

Discussion

A question/answer sequence is initiated by this command. Refer to Appendix E for a description of the responses that can be given in the question/answer sequence.

Examples

1) BCDASC CARDS;A.CARDS
   (converts the contents of the file CARDS to ASCII and stores the result in the file A.CARDS)

2) BCDA
   (prompts for the names of the BCD and ASCII files)

3) BCDASC B.DATA;*
   (converts the contents of the file B.DATA to ASCII and stores the result in the current file)

4) BCDA BCD.INFO
   (assumes the current file for the ASCII file)

5) BCDASC B.FILE;
   (prompts for the ASCII file name)

Purpose

To print in BCD the contents of an ASCII time sharing format file or files on the host high-speed printer.

Format

BPRI[NT][<ascfile>[;<ascfile>]...]

<ascfile>::= *|<filedesc>

Discussion

The contents of the ASCII file are converted to BCD before they are printed. This means that all lowercase letters are converted to uppercase and all control characters and some special characters are replaced with a blank. In addition, only the first 80 characters of each line are printed.

Fifty-two lines of the ASCII file are printed on each page. A one-line header is printed in the top margin of each page, consisting of the snumb, date, time and characters 16 through 75 of the first line of the file as the title and the page number. There is no recognition of any slew control that may be in the ASCII file.

BPRINT initiates a question/answer sequence if the file does not contain first-line reformatting information.

Since a batch job is spawned by this command, the batch $IDENT card information is requested from the user. See Appendix E for an explanation of the question/answer sequence.

BPRINT provides an easy way to list long files on a high-speed printer.

Examples

1) BPRINT (assumes the current file for the ASCII file)

2) BPRINT * (same as BPRINT)

3) BPRINT LONGFILE

4) BPRI MAIN;/SUBCAT/SUB1

Purpose

To punch the contents of an ASCII time sharing format file or files onto BCD cards at the host.

Format

BPUN[CH][<ascfile>[;<ascfile>]...]

<ascfile> ::= *|<filedesc>

Discussion

The contents of the ASCII file are converted to BCD before they are punched. This means that all lowercase letters are converted to uppercase and all control characters and some special characters are replaced with a blank. In addition, only the first 80 characters per line are punched. The characters are punched in the Honeywell character set which differs from the IBMF and the IBMEL character sets.

BPUNCH initiates a question/answer sequence if the file does not contain first-line reformatting information.

Since a batch job is spawned by this command, the batch $ IDENT card information is requested from the user.

BPUNCH provides a means of creating a hard copy backup (cards) for time sharing files.

Examples

1) BPUNCH (assumes the current file for the ASCII file)

2) BPUNCH * (same as BPUNCH)

3) BPUNCH DATA.CRD

4) BPUN MAIN;/SUBCAT/SUUB1

Purpose

To run a specified BASIC file.

Format

```
BRN[H][-<time>][ <filedesc>][=<objfil>[(NOGO)]]
```

Description

<time> is the maximum central processor (cpu) time in seconds that the program is allowed for execution.

<objfil> is the file where the resulting object code is to be saved. If this file does not already exist, it is created as a random file with no general permissions.

Discussion

The user can control two phases of a BASIC run. The first phase is compilation during which the BASIC source statements are translated into a code the computer can understand. The code is called the object code and it can be saved in a file as in Examples 5 and 6. Note, the object code cannot be modified directly by the user. If the program has to be changed, the BASIC source statements must be changed and a new object file created by running the source program again.

The second phase is execution during which the object code is loaded into memory and executed. A time limit on execution can be imposed as in Examples 3, 4 and 7. This time limit represents the actual processor time used for the execution which is considerably less than the elapsed time. It is a good idea to specify a time limit if the program is being executed for the first time or after substantial changes have been made. This will ensure that a limited amount of the computer resources are wasted if the program has a looping error.

Programs should be saved as object code if they are going to be CHAINed to repeatedly from within another program. This will eliminate the time and cost of recompiling each time the program is CHAINed to.

Examples

1)	BRN	(compiles and executes the current file)
2)	BRNH ANNUIT	(If file ANNUIT contains a BASIC source program, compiles and executes it. If file ANNUIT contains BASIC object code, executes it. The time and date are printed as a header.)
3)	BRN -10	(compiles the BASIC source program in the current file and executes it for no longer than 10 cpu seconds)
4)	BRN -10 ANNUIT	(similar to Example 2 except that it executes the program for no longer than 10 cpu seconds)
5)	BRN LIMITS=O.LIMITS	(assuming that file LIMITS contains a BASIC source program, compiles it, stores the object code in file O.LIMITS, and executes it)
6)	BRN STATS=O.STATS(NOGO)	(similar to Example 5 except that the program is not executed because of the NOGO option)
7)	BRN -10 O.STATS	(assuming that file O.STATS contains BASIC object code, executes it for no longer than 10 cpu seconds)
8)	BRN /MARTIN/GUITARS	
9)	BRN USERID/FILE,R	

Purpose

To sequence the line numbers of a BASIC program that is in the current file and modify the corresponding statement number references (such as GOTO, IF...THEN, etc.).

Format

```
BSEQ[UENCE][ [<initial>][,[<increment>][,<line-range>]]]

  <initial>      ::= <line>
  <increment>    ::= a decimal number
  <line-range>   ::= <begin-line>-<end-line>
                   |  <begin-line>[-]
                   |  -<end-line>
  <begin-line>   ::= <line>
  <end-line>     ::= <line>
  <line>         ::= a 1- to 8-digit decimal number
```

Description

<initial>	the 1- to 8-digit line number to be used as the first line number in the sequencing. The default value is 10.
<increment>	the increment between line numbers. The default value is 10.
<begin-line>	the 1- to 8-digit line number at which sequencing is to begin. If <begin-line> is not specified, the first line number of the file is assumed.
<end-line>	the 1- to 8-digit line number at which sequencing is to stop. If <end-line> is not specified, the last line number of the file is assumed.

Discussion

Care should be taken in sequencing concatenated BASIC files as the statement references may become invalid.

WARNING- The results are unpredictable if the current file does not contain a BASIC program.

Examples

1) BSEQUENCE (sequences all the line numbers and modifies the statement references, using 10 as the beginning line number and 10 as the increment)

2) BSEQUENCE 10,10
(same as BSEQUENCE)

3) BSEQ ,5 (sequences all the line numbers and modifies the statement references using 10 as the beginning line number and 5 as the increment)

4) BSEQ 2,2,-100 (sequences the line numbers from the beginning of the current file to the current line 100 using 2 as the new beginning line number and 2 as the increment, and modifies the statement references throughout the file)

5) BSEQ 500,10,300-
(sequences the line numbers from the current line 300 to the end of the current file, using 500 as the new beginning line number and 10 as the increment, and modifies the statement references throughout the file)

6) BSEQUENCE 500,10,300
(same as Example 5)

7) BSEQ 400,,400-600
(sequences the line numbers from the current lines 400 to 600 using 400 as the new beginning line number and 10 as the increment, and modifies the statement references throughout the current file)

Purpose

The BSP command backspaces a designated tape cassette one record.

Format

BSP TAPE [n]

[n] ::= 1 or 2, default value is 1.

Discussion

This command can only be used on 7700 Series VIP (device 13, 14, or 15 octal).

Purpose

To disconnect the terminal and to report the user's system usage in the following terms:

- dollars used during the current "LOGON" session
- dollars used to date during this billing period and the percent of the monthly allotment that amount represents
- the storage LLINKS in use, the total LLINKS allocated to the user-ID and the percent of those LLINKS that are in use.

Format

BYE

Discussion

Before the usage report is printed, the Available File Table (AFT) is scanned for user's temporary files. A message is issued as to the number of temporary files, after which the user is queried as to their disposition. Each temporary file name is printed followed by a question mark. The user will not be queried for a disposition of file ABRT if it is a temporary file. The user may respond as follows:

1) carriage return - implies the file is not to be saved as an entry in the file system.

2) NONE - implies this file and all of the succeeding temporary files are not to be saved.

3) SAVE[<filedesc>] - specifies that the temporary file is to be saved in the permanent file specified by <filedesc>. If the permanent file does not already exist, it is created with general READ permission.

Example

BYE

Purpose

The CARDIN command invokes the CARDIN subsystem to allow a terminal user to submit a job to the batch portion of the system. Job streams consist of intermingled data and control card images which are processed by batch job flow.

Format

```
CARD[IN] [<infile>]

  <infile>     ::= <filedescr>
```

Discussion

Unless the input file contains first-line formatting (example 1), the CARD subsystem responds to RUN with CARD FORMAT, DISPOSITION?

The response to FORMAT can be one of following:

```
ASIS   (run the job (file) as is)
or A

MOVE   (move sequence numbers to field
or M      numbers 73-80)

NORM   (implies the MOVE option and
or N     normal tab character (the colon)
         and tab settings (8, 16, 32, 73) have been
         employed in building the file)

STRIP   (remove line numbers and make
or S     the first nonnumeric the first
         character in each line)
```

The response to DISPOSITION can be one of the following (commas included as shown):

```
,JOUT      (save all implied files for examination
or, J       by the JOUT subsystem)

,ROUT(XX) (direct output to station XX)
or, R

,TALK      (enter conversational mode - must
or , T      have permission specified for the
            user's SMC from the MASTER user)

,URGC(XX) (assign specified urgency to this
or, U       job)

,WAIT      (wait for job termination)
or, W
```

If the response to FORMAT was A, M or S, (not N), the subsystem prompts:

TAB CHARACTERS AND SETTINGS?

If the user did not use any tabs in preparing his input, the response is a carriage return; otherwise, the response is the tab character followed by tab settings separated by commas.

;,4, 8, 16, 32, 73

Specifies that the tab character is a semicolon and tab settings are 4, 8, 16, 32 and 73.

Examples

1)
```
*CARD
*OLD CARDJOB
*LIST
10$       IDENT    cccc.iii,bbbnn,X0000
12$       OPTION   FORTRAN
14$       FORTY
16        READ,A,B,C
18        D = A + B + C
20        PRINT 20,D
22  10    STOP
24  20    FORMAT (4H D =, F10.2)
26        END
28$       EXECUTE
30$       LIMITS  01,10k,,2000
32#2,7,3,4,687
34$       ENDJOB
*RUN

   CARD FORMAT,DISPOSITION ?  STRIP,ROUT (AB)

   TAB CHARACTERS AND SETTINGS?
   * SNUMB 1234T
```

2)
```
*CARD
*OLD
OLD NAME - REACBIN

*LIST
10##NORM,ROUT(id)
20$:IDENT:cccc,iii,bbbnn,X0000
30$:OPTION:FORTRAN
40$;SELECT:FSxxxx/CSTAREAC      (Binary File)
50$:EXECUTE
60#215,365
70#216,37
80#6305,45
90#605,271
100#125,68
110$:ENDJOB
*RUN

  SNUMB 1224T
```

The double number sign in line 10 is interpreted by the CARD subsystem to mean that this record is first line formatting information. The number signs in lines 60 through 100 separate the line number from numeric data that is to be used during program execution.

3) The following sample program illustrates the use of interrelated time sharing subsystems and batch programming features. The program is submitted by means of the time sharing CARDIN subsystem. Direct conversation between the program and the user's terminal is then initiated, and use is made of two conversational batch features--Conversational Debug Routine (RBUG) and conversational I/O extensions to File and Record Control. Text within brackets is not part of the program but has been added to illustrate particular features.

The program, submitted under CARDIN, makes use of conversational I/O extensions to File and Record Control.

```
0100$;IDENT;VXE00,JDOE
0200$;OPTION;FORTRAN
201$,USE;.RTYP  Required when $ DAC cards are present.
0300$,FORTRAN;NDECK
400#2;WRITE(6,3)
0500#3;FORMAT(27HPROGRAM TO CALCULATE RECOIL)
0600#1;WRITE(6,4)
0700#4;FORMAT(9HRIFLE WT.)
0800;READ(5,5,END=999)WR
0900#5;FORMAT(V)
1000;WRITE(6,7)
1100#7;FORMAT(10HBULLET WT.)
1200;READ(5,5)WB
1300#9;WRITE(6,10)
1400#10;FORMAT(8HVELOCITY)
1500#11;READ(5,5)VB
1600;WRITE(6,13)
1700#13;FORMAT(10HPOWDER WT.)
1800;READ(5,5)WP
1900;X=WB*VB+4700.*WP
2000;Y=7000.*WR
2100;Z=WR/64.4
2200;E=Z*(X/Y)**2
2300#15;WRITE(6,16)E
2400#16;FORMAT(8HENERGY= ,F6.2,9H FT. LBS.)
2500;GO TO 1
2550#999;STOP
2600;END
2700$;EXECUTE
2800$;DAC;05
2900$;DAC;06
3000$;ENDJOB
```

The program is then formatted for legibility.

```
*PRINT
CARD FORMAT ?
NORM(;)

09/10/79    09.69
```

```
0100   $     IDENT    VXE00,JDOE
0200   $     OPTION   FORTRAN
201    $     USE      .RTYP
0300   $     FORTRAN NDECK
400    2     WRITE(6,3)
0500   3     FORMAT(27HPROGRAM TO CALCULATE RECOIL)
0600   1     WRITE(6,4)
0700   4     FORMAT(9HRIFLE WT.)
0800         READ(5,5,END=999)WR
0900   5     FORMAT(V)
1000         WRITE(6,7)
1100   7     FORMAT(10HBULLET WT.)
1200         READ(5,5)WB
1300   9     WRITE(6,10)
1400   10    FORMAT(8HVELOCITY)
1500   11    READ(5,5)VB
1600         WRITE(6,13)
1700   13    FORMAT(10HPOWDER WT.)
1800         READ(5,5)WP
1900         X=WB*VB+4700.*WP
2000         Y=7000.*WR
2100         Z=WR/64.4
2200         E=Z*(X/Y)**2
2300   15    WRITE(6,16)E
2400   16    FORMAT(8HENERGY= ,F6.2,9H FT. LBS.)
2500         GO TO 1
2550   999   STOP
2600         END
2700   $     EXECUTE
2800   $     DAC      05
2900   $     DAC      06
3000   $     ENDJOB
```

[The program is then passed to the batch system for processing; the TALK option permits direct-access connection.]

```
*RUN
  SNUMB # 0165T
CARD FORMAT, DISPOSITION ?
NORM(;),TALK
PROGRAM TO CALCULATE RECOIL
RIFLE WT.
05?8.5
BULLET WT.
05?150
VELOCITY
05?3200
POWDER WT.
05?58
ENERGY=320.06 FT. LBS.
RIFLE WT.
05?            Carriage return; null response.
CLOSING FILE 05
CLOSING FILE 06

ACTIVITY TERMINATED
NORMAL TERMINATION
```

A $ USE RBUG card is substituted for $ USE .RTYP to initiate the RBUG subroutine, and breakpoints are inserted. The program is then formatted in its new version.

```
*201$;USE;RBUG
*202$;DUMP; .....
*203;DEBUG;2/(BREAKPOINT)
*204;DEBUG;15/(BREAKPOINT)
*PRINT
CARD FORMAT ?
NORM(;)

09/10/79    09.85

0100    $    IDENT    VXE00,JDOE
0200    $    OPTION   FORTRAN
201     $    USE      RBUG
202     $    DUMP     ......
203          DEBUG    2/(BREAKPOINT)
204          DEBUG    15/(BREAKPOINT)
0300    $    FORTRAN NDECK
400     2    WRITE(6,3)
0500    3    FORMAT(27HPROGRAM TO CALCULATE RECOIL)
0600    1    WRITE(6,4)
0700    4    FORMAT(9HRIFLE WT.)
0800         READ(5,5,END=999)WR
0900    5    FORMAT(V)
1000         WRITE(6,7)
1100    7    FORMAT(10HBULLET WT.)
1200         READ(5,5)WB
1300    9    WRITE(6,10)
1400    10   FORMAT(8HVELOCITY)
1500    11   READ(5,5)VB
1600         WRITE(6,13)
1700    13   FORMAT(10HPOWDER WT.)
1800         READ(5,5)WP
1900         X=WB*VB+4700.*WP
2000         Y=7000.*WR
2100         Z=WR/64.4
2200         E=Z*(X/Y)**2
2300    15   WRITE(6,16)E
2400    16   FORMAT(8HENERGY= ,F6.2,9H FT. LBS.)
2500         GO TO 1
2550   999   STOP
2600         END
2700    $    EXECUTE
2800    $    DAC      05
2900    $    DAC      06
3000    $    ENDJOB
```

The program is again passed to the batch system, along with the TALK option. Control of the program is obtained at breakpoints, interrogations are made, and the program is then permitted to continue and run to termination.

```
*RUN
  SNUMB #0166T
CARD FORMAT, DISPOSITION ?
NORM(;),TALK
***ROUTINE  ......  LOC 2           COUNT    000001
???R
PROGRAM TO CALCULATE RECOIL
RIFLE WT.
05?8.5  8 AND 1/2 POUNDS
BULLET WT.
05?150.0  150 GRAINS
VELOCITY
05?3200.0   FEET PER SECOND
POWDER WT.
05?58.0  GRAINS
***ROUTINE  ......  LOC 15          COUNT    000001
???FE  GET A PEEK AT ANSWER
E
???R  LOOKS GOOD
ENERGY= 21.11 FT. LBS.
RIFLE WT.
05?7.5
BULLET WT.
05?150.
VELOCITY
05?2175
POWDER WT.
05?0.  AGAIN FORGOT THE DECIMAL POINT
***ROUTINE  ......  LOC 15          COUNT    000004
???R#2  TRY AGAIN
***ROUTINE  ......  LOC 2           COUNT    000004
???R    TRY A "NORMAL" RUN
PROGRAM TO CALCULATE RECOIL
RIFLE WT.
05?7.5
BULLET WT.
05?150.0
VELOCITY
05?2175.0
POWDER WT.
05?31.0
***ROUTINE  ......  LOC 15          COUNT    000002
???FE   PEEK AT ANSWER
E        0.94112817E 01
???R  ANSWER SEEMS ABOUT RIGHT FOR 30/30 WITH LIGHT LOAD
ENERGY=   9.41 FT. LBS.
```

```
RIFLE WT.
05?7.5          TRY 30.06 TYPICAL LOAD
BULLET WT.
05?180.0
VELOCITY
05?2505.0
POWDER WT.
05?45.0
***ROUTINE  ......  LOC 15          COUNT     000003
???R     LET IT GO NORMALLY
ENERGY=  18.54 FT. LBS.
RIFLE WT.
05?0.
BULLET WT.
05?0.
POWDER WT.
05?0.
       DIV CHECK     AT LOCATION   037651
      EXP OVERFLO    AT LOCATION   037400
***ROUTINE  ......  LOC 15          COUNT     000004
???T LET THE PROGRAM QUIT NORMALLY
**EXIT
CLOSING FILE 05
CLOSING FILE 06

ACTIVITY TERMINATED
NORMAL TERMINATION

*BYE
```

Purpose

The CATALOG command lists the names of subcatalogs and files which are under the current "LOGON" user-ID or to list the attributes of a specified file in the current "LOGON" user-ID.

Format

```
CATA[LOG] [&lt;filedesc&gt;|&lt;catdesc&gt;|#CMD|#LIB|]
          &lt;catdesc&gt;&lt;options&gt;|&lt;user-ID&gt;**
```

#CMD:: produces a catalog listing for the CMDLIB user-ID. The CMDLIB user-ID contains user supplied routines which can be used to augment the common command list.

#LIB:: produces a catalog listing for the LIBRARY user-ID. The LIBRARY user-ID contains various routines in a user accessible library.

<options>:: n|x(mm-dd-yy)|S|A|R|FIRST/name/|*

Allows limited listing of catalog and file names. Optional fields are:

x, to specify whether the date is date created (C), date of last access (A), or last date the file was changed (L)
date, in the form mm-dd-yy is required when C, A, or L is requested.
n, number of files to be listed, starting from the oldest to most recent
R, reverse the list before printing
S, sort the names
A, list in abbreviated fashion (eight per line)
*, Prints a detailed list of catalog attributes

Any option may be omitted and the order given is immaterial. The FIRST/name/ allows the cat/file list to start at the specified name.

<user-ID>**:: Lists catalog and file names emanating from the UMC of the specified USERID. The requesting user must have modify permission at the UMC level of the specified USERID.

Discussion

The CATALOG command can normally only reference the names of subcatalogs and/or files that are in the current "LOGON" user-ID.

Passwords on subcatalogs and files do not have to be specified when they are referenced in the CATALOG command.

In Examples 3 and 4, the attributes for file SMITH and file UNITS, respectively, will be listed. The attributes that are listed for a filename are:

File Name	Current File Size In Blocks
Originator	File Type
Date Created	Device
Date Changed	General Permissions
Last Date Accessed	Specific Permissions
Max File Size In Blocks	

In Examples 1 and 2, the names of all files and subcatalogs that are directly under the current "LOGON" user-ID are listed. But, the names of any files and/or subcatalogs that are under a subcatalog which is directly under the current "LOGON" user-ID are not listed by this form of the command. In order to list the names of all files and subcatalogs that are under another subcatalog, all of the subcatalog levels to the desired file must be specified in the <filedesc> of the CATALOG command.

In order to list the names of all files and subcatalogs that are under a user-ID, a CATALOG command must be issued for each catalog.

In the listing from a CATALOG command, to distinguish a file name from a subcatalog name, file names are indented two columns to the right of subcatalog names. Note, that file names listed after a subcatalog name do not belong to that subcatalog. Example 5 lists the names of all files and subcatalogs that are under subcatalog METRIC, whereas Examples 1 and 2 list the names of all files and subcatalogs that are directly under the current "LOGON" user-ID.

For an explanation of the file system with a description of subcatalogs and files in a user-ID, refer to File System in Section II.

A user who is the originator of (or has modify permission for) a catalog belonging to another user may list, purge, or release the catalog. This also applies to a specific list request for a catalog or file. Permission is not required to list strings which originate from the user's own catalog, the library (#LIB), or the command library (#CMD). Passwords need not be given in the catalog structure unless the specified file or catalog was created by another user.

Examples

1) CATALOG
(lists the names of the subcatalogs and files that are directly under the current "LOGON" user-ID)

2) CATA
(same as CATALOG)

3) CATA SMITH
(lists the attributes of the file SMITH)

4) CATA /METRIC/UNITS
(lists the attributes of the file UNITS which is under the subcatalog METRIC)

5) CATA /METRIC
(lists the names of the subcatalogs and files that are under subcatalog METRIC)

6)
```
*CATA,A,S
LIST OF CATALOG TSSUNIT ON 10/12/79 AT 10.786

*4JS2      4JS2DOC   4JS2MAC *4VX       5KDOC    *7.1      A
 APNT      APRI      ARH1    *ARH       ARHTMP    ASM-EX    B
 DATETIME  DATTIM    DICTION  DRLT      DRLTST    DRLTST-S  DUMMY
*F1        FILED     G       *HAUGH    *HOLDRIDG  ITS-XMIT*KNUDSON
 MAIL.BOX  MAIL.BX   MAK4JS2X MAKE4JS2 MAK.MAC1  MAK.MAC2  MERG.MAC
*MILLER    MMDUMP    MONTHLY  PGTST     PGTST-S   PP-UNUO   QSTAR
 RESTORJ2  SAVEMAIL*SCHOFFEL  SS4JS2    SS4JS2.X  SY        TCTMP
 TEMPA     TEMPE     TEMPJCL  TEMPQ     TEST      TESTSRC   TS1
 TSRT     *TSSDOC   *TURNEY

*CATA/TSSDOC

    LIST OF CATALOG TSSDOC ON 10/12/79 AT 10.794
    CATALOGS
      FILES

      GENINFO
      TERMBATC
      SYSPROG
      TEXTEDIT
      BASIC
      MASTER
      TRACE
      FORTRAN
      CRUN

*CATA /TSSDOC,A,S,R

    LIST OF CATALOG TSSDOC ON 10/12/79 at 10.799

TRACE    TEXTEDIT TERMBATC SYSPROG  MASTER   GENINFO  FORTRAN
BASIC

*CATA /TSSDOC/TRACE

    FILENAME-TRACE
    ORIGINATOR-TSSUNIT
    DATE CREATED-040279
    DATE CHANGED-040279(10.509)
    LAST DATE ACCESSED-040279
    NUMBER OF ACCESSES-2
    MAX FILE SIZE-3620 LLINKS
    CURRENT FILE SIZE-181 LLINKS
    FILE TYPE-LINKED
    DEVICE-ST3
    GENERAL PERMISSIONS-NONE
    SPECIFIC PERMISSIONS-NONE
```

Purpose

The CMOD command allows the user to modify or display the Program Switch Word (PSW).

Format

```
CMOD operation-1;operation-2;...;operation-n
```

Discussion

Each operation is an arithmetic, Boolean operation, or action item and when evaluated affects the lower half (bits 18-35) of the PSW.

An operation may include any of the following forms:

	MEANING	RESTRICTIONS
n	Set C(PSW) bits 18-35 to n	$n<262144$
+n	Add n to C(PSW) bits 18-35	$n<262144$
-n	Subtact n from C(PSW) bits 18-35	$n<262144$
Si	Set bit i	$18\leq i< 36$
Si-j	Set bits i through j	$18\leq i\leq j <36$
Ri	Reset bit i	$18\leq i< 36$
Ri-j	Reset bits i through j	$18\leq i\leq j<36$
Ni	Negate bit i	$18\leq i<36$
Ni-j	Negate bits i through j	$18\leq i\leq j<36$
D	Display C(PSW) bits 0-35	
E	Store error code in PSW bits 18-35	

Examples

1) CMOD 4096
   (set the octal equivalent of 4096 in the PSW).

2) CMOD +4096
   (add the octal equivalent of 4096 to the PSW).

3) CMOD -4096
   (subtract the octal equivalent of 4096 from the PSW).

4) CMOD S18, S19, S20-23
   (set bits 18 and 19 and 20 through 23).

5) CMOD R22, R30-35
   (reset bits 22 and 30 through 35).

6) CMOD N22, N30-35
   (negate bits 22 and 30 through 35).

7) CMOD D
   (display the 36-bit contents of the PSW

8) CMOD E
   (store the error code in the PSW bits 18-35).

Purpose

The CODE command is used to encrypt a file, making its content unintelligible to others.

Format

```
CODE[<infile>[;<outfile>]]

<infile>  ::= <filedesc>

<outfile> ::= <filedesc>
```

Discussion

Input and output file descriptions may accompany the command, separated from one another by a semicolon. The input file may be random or sequential (* designates the current file) and either description may include passwords, permissions, an alternate name and up to eight levels of subcatalogs. The output file will be created for the user if it does not already exist.

Upon receipt of the CODE command, the user is asked to enter (and verify) a key to be used for encrypting the file. This is simply a one- to 12-character password which may include any combination of uppercase and lowercase alphabetics and nonprinting characters such as BELL, TAB, SPACE, etc. The encryption key is known only to the user; i.e., it is neither resident on mass storage nor retained anywhere in the system.

DECODE is used to obtain the original version of an encrypted file. Input and output file descriptions are required as for CODE and are managed in a similar manner. The output file may be * to designate the current file. Upon receipt of the DECODE command, the user is asked to enter the encryption key. This is the same key originally used with the CODE command to encrypt the file.

File encryption offers a level of data base security unsurpassed by other methods presently available for privacy protection. Sensitive information can be safely stored in the system without fear of penetration by unauthorized individuals. Even if a penetrator obtains the coded file and knows the encryption algorithm used to produce it, the file <u>cannot</u> be decoded except by trial and error -- a process obviously prohibitive by the 12-character key length. Note that it is, of course, necessary to destroy the original file once it has been encrypted. This may be conveniently accomplished with post use of the PURGE or ERASE command.

Examples

1) CODE MYFILE;URFILE

2)
```
*CODE A;TEMP
KEY:
[redacted]       (HI)
*CODE A;TEMPCODE
KEY:
[redacted]       (HI)
VERIFY:
[redacted]       (HI)
*FDUM TEMPCODE    (VERIFY ENCRYPTION)
BLOCK TO BE READ? 1 SO-47
000000  031616255014   262273434670   577215303777   363363370564

000004  416563126332   722751071744   205642217313   057516177520

000010  501451300263   676263070033   557256613605   560034215716

000014  536131422027   025425567775   340760565060   362214472422

000020  223034766511   720054565221   274631407044   672110313037

000024  401323415111   455217354156   471112520351   443434635471

000030  261012513313   704356700110   725535234102   535456401613

000034  411333264333   226031124476   663730167360   641546120142

000040  610313010536   707514417440   462036746470   653573061613

000044  034061237110   107255640411   217066204543   463172740062

? D
*DECO TEMPCODE;*
KEY:
[redacted]       (HI)
*LIST

1
2
3
4
```

Purpose

The CONNECT command allows a user that was unintentionally disconnected to call back in and reconnect to the same session and User Status Table (UST) previously used. The resumption of a previous session means that the User Status Table (UST) entry for the disconnected user is held until either the site specified reconnect time expires or the reconnection is made.

Format

CONN[ECT] [channelno]

[channelno]::= (Channel number as it appears in the sign on message).

Discussion

An accidental disconnect may be caused by noise on the phone line, accidentally disrupting the data set, accidently hitting the clear key, or accidently hitting control-C. Whenever a premature disconnect occurs the disconnected user may attempt to reconnect to the previous session, provided the site-defined time out has not passed. When a user knows his USERID is unique and that no other session is ongoing with that USERID, the line number does not have to be included in the CONNECT attempt. In addition, a provision has been made to allow a user to CONNECT to a session on another copy of TSS (multicopy TSS option).

Examples

```
0110401

HIS TIMESHARING ON 01/30/77 AT 5.241 CHANNEL  2140 TS1

USER ID- SMITH
PASSWORD--
ⱡⱡⱡⱡⱡⱡⱡⱡⱡⱡ
OLD TEST
*STAT
CHANNEL 2140 TS1
USER STATUS ON JAN 30, 1977 AT 5:14:51 LOG-ON AT 5:13:53
PROC TIME USED 0.00 SEC., 1 FILE I/O 80 CHAR KEY I/O
LIST OF OPEN FILES : TEST

*LIST
10 THIS IS A TEST
(line drops)
```

user re-dials

```
0110401

HIS TIMESHARING ON 01/30/77 AT 5.257 CHANNEL 2150 TS1

USER ID - SMITH$JOHN
*CONNECT 2140
10  THIS IS A TEST
20  OF THE RECONNECT     OUTPUT CONTINUES
30  FEATURE              FROM PREVIOUS LIST
40  END
* STATUS
CHANNEL  2150 TS1
USER STATUS ON JAN 30, 1977 AT 5:15:51 LOG-ON AT 5:13:53
PROC TIME USED 0.05 SEC., 2 FILE I/O 264 CHAR KEY I/O

LIST OF OPEN FILES:  TEST

*
```

Purpose

The CONVERT command invokes the CONVERT subsystem to perform file manipulation of text between files.

Format

```
CONV[ERT][infile(s)][=otfile][:options]

     [infile(s)]  ::= *|[filedesc]
     [=otfile]    ::= *|*SRC|[filedesc]
     [:options]     as described below.
```

Description

Media Code Options

The output record format options specify the physical format of the output record. The default option for the CONVERT command is "ASCII". The options and their meanings are as follows:

BCD - variable-length BCD - media code 0

COMDK - BCD compressed deck card image (COMDK) - media code 1

CARD - BCD 14-word card image - media code 2

PRINT - BCD variable-length print line image - media code 3

OLDASC - obsolete TSS ASCII - media code 5

ASCII - standard system format ASCII - media code 6

APRINT - ASCII print line image - media code 7

ACARD - ASCII card image - media code 10

SAME - a record output media code is the same as its input media code

Line Number Options

Line numbers can exist with COMDK, CARD, ACARD, OLDASC, and ASCII records. All BCD, PRINT, and APRINT records cannot possess line numbers. The line number for an ASCII or OLDASC record consists of 1 to 8 numeric characters. These numeric characters must be among the first eight characters in a line. A line number is defined to include any leading blanks. A line number is terminated by a non-numeric character, including blank. If the "#" character terminates a line number and, if it is one of the first eight characters of a line, it is considered to be a delimiter; it is treated as neither part of the line number nor part of the text. The line number for COMDK, CARD, and ACARD records is defined to be all the trailing digits in columns 73-80. This field may begin with non-numerics; these also are considered neither part of the line number nor part of the text.

The line number options may specify:

1. Whether line numbers are to appear in the output text.

2. The actual line number values.

The default line number option is "ASIS". A description of each of the options follows:

ASIS — Line numbers are assumed not to be present in the input file. Text, including leading/trailing numeric characters and "#"'s are left as is.

STRIP — Strip line numbers from the input text before reformatting and writing the output text. Input COMDK, CARD, and ACARD records are truncated at column 72. Line numbers on ASCII and OLDASC records, when present, are discarded and the first character following the line number is treated as the first character of the line.

MOVE — Move line numbers. The input records have the line numbers detached from the text string, either from the front (ASCII or OLDASC) from columns 73-80 (COMDK, CARD, or ACARD). The output records have the line numbers re-attached to the text string, either at the front (ASCII or OLDASC) or in columns 73-80 (COMDK, CARD, or ACARD). If the output records are BCD, PRINT, or APRINT, the line numbers are not re-attached and the M option acts similar to the S option.

I(i,j) — Insert line numbers beginning with line number i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. The input file is assumed not to be line-numbered. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the I option is ignored.

R(i,j) — Resequence line numbers. Strip any existing line numbers from the input text and insert new line numbers in the output text, beginning with i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the R option behaves as the S option.

N(ch) — Implies the M option and specifies that the normal tab character (the colon) and tab settings (8, 16, 32, 73) have been employed in building the input file(s). The (ch) argument may be used to define a character which replaces the colon as the tab character.

LABEL (abcde(i-j)fghij(i-j)---) If the output records are COMDK, CARD, or ACARD, then a label is placed left-justified in columns 73-77. The label is specified as 1 to 5 non-blank characters. The fields "abcde" and "fghij" represent the labels. The label is placed on only those lines with line numbers between i and j inclusive. Up to 10 distinct labels may be given. If more than one label is given though, the (i-j) specifications may not overlap.

The LABEL option is meaningful only if line numbers are attached to output records. Therefore, the label option is completely ignored unless it is accompanied by either the insert, resequence, or move option.

For the I and R options, output line numbers for ASCII and OLDASC records will have at least the number of digits specified for i in I(i,j) or R(i,j). Thus R(0010,10) will result in line numbers 0010, 0020, 0030,---.

Input records are assumed to have line numbers when the STRIP, MOVE, and RESEQUENCE options are specified. Otherwise, line numbers are assumed to be absent and leading numerics in ASCII format are treated as real text. When line numbers are assumed present, tabbing and columnizing are performed relative to the start of the real text.

The user must be careful not to alter the line number values of a BASIC file.

Character Manipulation Options

A description of each of the character manipulation options follows.

TAB(ch,i,j---;ch,i,j---;----) Expand tab characters into blanks. Use "ch" as a tab character with settings i,j,k,etc. Usually, any occurrence of the tab character in the input file(s) results in the replacement of the tab character with a string of blanks up to the next tab setting. However, if a tab character is encountered beyond the last tab setting specified for that tab character, it is treated as a normal nontab character.

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the tab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

UNTAB(ch,i,j---;ch,i,j---;----) Insert tab characters, replacing blanks. Use "ch" as a tab character with settings i, j, k, etc. Any occurrence of a string of blanks terminating on an "untab" tab stop is replaced by the character "ch".

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the untab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

LOWER Convert all alphabetic characters to lowercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

UPPER — Convert all alphabetic characters to uppercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

BEGIN(ch) — Begin a new line (record) immediately after the character "ch". The character "ch" is treated as a delimiter and not part of the text. It is not placed in the output text. When the "ch" character is located at the beginning or end of a line, it is simply deleted. Strings of the "ch" character are treated as a single "ch" character.

COLUMNS(i-j) — Delete all of the characters in a line except those which are located within columns i through j inclusively. The options BEGIN and TAB are both completed before COLUMNS takes effect. If a record does not extend through column j prior to the COLUMNS option execution, it is blank-filled to column j. Thus, when the COLUMNS options is in effect, the length of all generated output records is j-i+1 characters.

SQUEEZE — Replace any string of two or more blanks by a single blank. The options BEGIN, TAB, COLUMNS, and UNTAB are all performed before SQUEEZE is executed.

TRAIL — Delete all trailing blanks on a line. The TRAIL option is performed immediately after the SQUEEZE option.

A number of options affect the length of an output text line. It is important that the user adhere to the order in which these options are performed. The order (from first to last) in which the options are executed is:

BEGIN
TAB
COLUMNS
UNTAB
SQUEEZE
TRAIL

Miscellaneous Options

VERIFY — If the VERIFY option is in effect when CONVERT completes the processing of an input file, then CONVERT gives a brief summary of the number of records obtained from the file. This summary gives, for each media code, the number of records which had that media code.

IGNORE — Ignore all embedded $$ control lines. Treat them as text.

DISCARD — Discard all nontext records. Nontext records are those records whose media code is not one recognized and interpreted by CONVERT. The JRN, JPRINT, JPUNCH, APRINT, and DISPLAY commands require that nontext records be discarded. The CONVERT command normally does not require that nontext records be discarded. When nontext records are encountered during the execution of the CONVERT command, they are written to the output file, but no reformatting or media conversion is performed.

TIME When the TIME option is invoked, the date and time of day are printed at the user's terminal.

DEFAULT The DEFAULT option is used to nullify all options which the user has specified either on the command line or embedded $$ control lines. The default option has no affect on any of the "specialized" options. Because of the nature of the DEFAULT option, it is meaningless for it to be located in the options field of the command line. Therefore, if the DEFAULT option is encountered in the options field, an error message is issued. The same reasoning applies to the placement of the DEFAULT option anywhere other than the beginning of a $$ control line.

File Processing Options

SELECT (file) The SELECT option is analogous to the $ SELECTA card. The select option allows an input file to specify other input files. Upon encountering the SELECT option, the selected file is obtained and is used in place of the $$ control line. Nesting of selects is permitted up to 17 levels. The SELECT option is meaningful and valid only on a $$ control line. Only one SELECT option may be specified on a $$ control line.

INCLUDE If the INCLUDE option is in effect, CONVERT, upon encountering the SELECT option, uses the selected file as an input file.

EXCLUDE If the EXCLUDE option is in effect, CONVERT ignores the SELECT option.

The INCLUDE and EXCLUDE options allow the user to control the performance of the select options while not forcing him to disregard:

1. Other options on the same $$ control line.

2. All $$ control lines.

The INCLUDE option is the default option for the JRN command. The EXCLUDE option is the default option for the JPRINT, JPUNCH, APRINT, DISPLAY, and CONVERT commands.

Specialized Options

The "specialized" options are a class of options completely distinct and separate from all preceding options. The "specialized" options are unlike other options in that they take effect only when all input files have been read, converted, and closed; i.e., after the output file has been completely generated. All other options, of course, are meant to be used when the output file is in the process of being generated.

JOUT — The JOUT option is applicable only to the JRN command. This option results in all implied files being saved so that they may be examined using the JOUT subsystem.

ROUT(xx) — The ROUT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. This option causes the implied files generated by the program execution to be directed to the specified two-character remote station. Only one ROUT entry is permitted.

WAIT — The WAIT option is applicable to the JRN, JPRINT, APRINT,and JPUNCH commands. This option causes the user to wait until the completion of the spawned job in the batch environment. The wait period may be broken out of by hitting the break key. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

TALK — The TALK option is applicable only to the JRN command. This option implies that the batch job includes execution of a program containing conversational (direct access) input/output. This option causes the user's terminal to be placed in direct access connection with the submitted program (by SNUMB) following its submission to the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

URGC(xx) — The URGENCY option is applicable only to the JRN command. This option indicates that the user wishes to assign initial urgency xx to the spawned batch job. If the assigned urgency is greater than the maximum allowed for the user, the message ILLEGAL URGENCY is sent and the batch job is not spawned. If xx is not specified, maximum allowable urgency is automatically assigned.

COPY(nn) — The COPY option is applicable only to the JPRINT, APRINT, and JPUNCH commands. This option causes the generation of nn multiple copies of the listing or punched deck. The maximum number of copies that can be produced from a single JPRINT/JPUNCH job is 13.

IDENT(info) — The IDENT option is applicable to the JPRINT, APRINT, and JPUNCH commands. This option allows the user to minimize the subsystem/user interface involved in the use of the JPRINT/JPUNCH commands. When the IDENT option is present, the normal question/answer sequence of

```
$ IDENT? response
```

is bypassed. The information presented as the IDENT option argument is used instead of the user-response to the question.

MONITOR — The MONITOR option is applicable to the JPRINT, APRINT, JPUNCH, and JRN commands. This option allows the user to monitor or track the status of a spawned job as it is executed in the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

DIRECT	The DIRECT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. If the DIRECT option is given on the command line, it overrides any JOUT or ROUT option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the JOUT option to place it on a $$ control line and later override it without changing the $$ control line.
DISMISS	The DISMISS option is applicable only to the JRN command. If the DISMISS option is given on the command line, it overrides any TALK, WAIT, or MONITOR option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the MONITOR option to place it on a $$ control line. He can then override it without being required to change his $$ control line.
INDENT(99)	The INDENT option applies to the APRINT command and specifies the number of print columns to indent from the left margin.
PAGELENGTH(99)	The PAGELENGTH option applies to the APRINT command and specifies the print page size for non-RUNOFF files being processed.

The ROUT, JOUT, and DIRECT options are mutually exclusive. The MONITOR, TALK, WAIT, and DISMISS options are also mutually exclusive. Mutually exclusive options are a group of options for which only one member of the group of options may be in effect. If the user attempts to give two mutually exclusive options in the options field of the command line or on a $$ control line, an error message is given.

Discussion

The CONVERT subsystem converts textual information from any text file format to any other text file format.

Examples

1) Retrieve the contents of saved file named FILEA and write contents onto the current file.

   CONVERT FILEA

   The text of FILEA is copied as is onto the current file in ASCII format. Line numbers, if present, are treated as text. There is no case shifting, tabulation, or reporting. This usage is equivalent to OLD FILEA except that non-ASCII files are accepted.

2) Save the contents of the current file on file named FILEA.

   CONVERT = FILEA

   The output file format is ASCII. Line numbers, if present, are copied as text. This usage is equivalent to RESAVE FILEA, except that if FILEA does not exist, CONVERT will create a temporary file of that name.

3) Save the current file on FILEA in card format.

CONVERT = /FILEA: CARD

If FILEA does not exist, CONVERT will create it as a permanent file.

4) List the contents of FILEA on the terminal.

CONVERT FILEA = **

This usage is identical to LIST FILEA except that non-ASCII files are accepted.

5) Resequence line numbers on the current file.

CONVERT: R(0020,20)

This usage is equivalent to RESE 20,20 (except for BASIC files). Minimum width of each line number is four digits.

6) Insert line numbers on the current file.

CONVERT: I

This usage is equivalent to RESE#.

7) Convert alphabetics on the current file to lowercase.

CONVERT: LOWER

8) Replace tab characters : and on the current file by the indicated number of blanks.

CONVERT T(:;>,7,13),M

The M option is required to cause tab stops to be calculated relative to the text and not relative to the beginning of the line numbers. Because no tab settings are specified for the : character, 8,16,32,73 are assumed.

9) Determine how many lines are on the current file.

CONVERT: V

10) Retrieve all lines between the 15th and 44th inclusively on file named SOURCE and insert the : tab character, with default settings of 8,16,32,73. Copy to the current file (the assumption is that SOURCE has no line numbers).

CONVERT SOURCE (#15-44):U

11) Save the lines numbered 10 to 100 from CATA/FILEA and lines 150 to 200 from the current file on file named SAVEFILE in card format, without line numbers.

CONVERT CATA/FILEA(10-100);*(150-200)=SAVEFILE:CARD,S

12) Save the current file on FILEA in card format with line numbers moved to columns 73 to 80 and with the label ABC on those lines with line numbers between 10 and 100 inclusive.

```
CONVERT = F LEA: CARD, M, L (ABC(10-100))
```

13) Concatenate the contents of files FILEA, FILEB, and FILEC, strip all line numbers, expand tabs, save in BCD variable-length records on SAVEFILE, and report the results.

```
CONVERT FILEA;FILEB;FILEC=SAVEFILE:S,V,BCD,

MORE? T(:,10,20,30,40,50)
```

Purpose

The CPY command copies the contents of one file into another.

Format

```
CPY[ <infile>;<outfile>[;<size>]]

<infile>    ::= *|<filedesc>
<outfile>   ::= *|<filedesc>
<size>      ::= a decimal number
```

Description

<infile> the file that is to be copied from.

<outfile> the file that is to be copied onto. If this file does not already exist, it is created to the correct size to hold all the data up to the first end of file and it is given general READ permission.

<size> the number of LLINKS to be copied, starting from the beginning of the file. The default is the number of LLINKS required to copy all the data up to the first end of file.

Discussion

Any type of file can be copied: ASCII, BCD, H*, etc.

The input file is not changed by the copy function.

This command can be used to effectively shrink a file, by copying a file whose content size is larger than its used size, into a new file. The system will create the new file just large enough to hold the logical file. The creation of an output file by the system preserves the incoming file's attributes (e.g. sequential versus random). The old file can then be RELEASEd and the new file renamed if desired. The user can copy part of a file by specifying the number of LLINKS that are to be copied starting from the beginning of the file. This could be used to recover part of a file that was destroyed by a system malfunction. The resulting file may not have an EOF (end-of-file mark), but it can be patched using the FDUMP subsystem.

Examples

1) CPY (prompts for the name of the input and the output files).

2) CPY TEMP;S.RECOVR
(copies the contents of the file TEMP onto the file S.RECOVR)

3) CPY DATA1;/PROJECT/JUN75
(copies the contents of file DATA1 onto the file JUN75 in subcatalog PROJECT)

4) CPY STATE BUDGET COSTS
(copies the contents of file BUDGET in user-ID STATE onto the file COSTS in the current "LOGON" user-ID)

5) CPY XX510001/MONDAY"X";MONDAY
(copies the contents of the file MONDAY in user-ID XX510001 onto the file MONDAY in the current "LOGON" user-ID. Since two files with the same name cannot be referenced at the same time, it is necessary to use an alternate name ("X" was used here) for the first one.

6) CPY PROG;REPAIR;2
(copies the first 2 LLINKS of the file PROG onto the file REPAIR. The file REPAIR may end up without an EOF.)

Purpose

The COUT command permits all output accumulated since either the beginning of a command file application or a previous COUT to be directed to the user's remote device or permanent file.

Format

```
COUT   *|<filedescr>[EXCLUDE]

[EXCLUDE]  ::= (eliminate user lines containing responses from the output)
```

Discussion

COUT is used in conjunction with command file or deferred processing.

Examples

```
COUT  * (all accumulated output is directed to the current file)

COUT FILEA (all accumulated output is directed to FILEA)

COUT FILE1;EXCLUDE
```

 CPOS

Purpose

The CPOS command permits either conditional alteration of the normal serial processing of input responses for a command file application or conditional execution of a single command, based on the contents of the PSW (Program Switch Word)..

Format

CPOS expression; operation.

Discussion

The first parameter represents a Boolean expression which, when true, causes the operation implied by the second parameter to be performed. Expression operands may include any of the following:

Form	Is True When:	Restrictions
i	Bit i of the PSW is on	$0 \leq i < 36$
LTn	Bits 18-35 of C(PSW) are less than n	n < 262144
LEn	Bits 18-35 of C(PSW) are less than or equal to n	n < 262144
GTn	Bits 18-35 of C(PSW) are greater than n	n < 262144
GEn	Bits 18-35 of C(PSW) are greater than or equal to n	n < 262144
EQn	Bits 18-35 of C(PSW) are equal to n	n < 262144
NEn	Bits 18-35 of C(PSW) are not equal to n	n < 262144
DEF	session is a deferred run	

Permissible operators include + (OR), * (AND), - (EXCLUSIVE OR) and / (NOT). Although / is a unary operator involving only one operand, by convention A/B is taken to mean A*/B. The expression is evaluated from left to right (without regard to operator hierarchy) and the resulting truth value (TRUE or FALSE) displayed. When false, or if a requested operation does not accompany the expression, CPOS processing is terminated. Otherwise, the operation is performed, constituting either a branch to a given response line on the input file or execution of a single specified command. A branch declaration is identified by an integer offset, optionally prefixed by + or - to denote forward or backward positioning (absence of the sign prefix implies forward positioning) or a label prefixed by a dollar sign. The offset represents the number of the response lines to forward or backspace, relative to the line containing the offset itself. Thus, the interpretation of the command,

```
CPOS  18+GE5*LT10;+2
```

is, "when bit 18 of the PSW is on or the magnitude of its lower half is greater than or equal to 5 and less than 10, then skip the command immediately following the CPOS". The expression itself is optional and, when omitted, causes the requested operation to be unconditionally performed; e.g. CPOS ;+2.

When an offset declaration requesting a forward space exceeds the number of response lines remaining, the application is terminated normally. Thus, the command,

```
CPOS EQ0;9999
```

results in terminating the command file application in which it appears when the lower half of the PSW is zero.

The requested operation may instead be a command to be conditionally executed, optionally accompanied by any necessary parameter declarations. For example,

```
CPOS  34-35;CRUN INFIL;OTFIL
```

would initiate the requested CRUN when either bit 34 or 35 (but not both) of the PSW is set.

Both CMOD and CPOS request continuation input when the last character of the line is a semicolon. A null response to such a request indicates completion. When parameters do not accompany either command (on the same line), the query, "FUNCTION?", is issued. The user must, at this time, enter all information necessary to perform the required operation.

Purpose

The COBOL-74 time sharing system provides the capability for compiling, loading, and executing COBOL-74 and I-D-S/II programs from a terminal under the control of the time sharing executive. Additionally, DM-IV Transaction Processing Routines may be compiled using this facility.

Format

```
CRN  [ { *                  }  ]
       { filedescriptor-1   }

     [ : [  IDENT=string-1   ]

      [;NOLOAD  ]

      [;NOGO  ]

     [ ;P*= { ONLINE           } ]
            { REMOTE(ii)       }
            { filedescriptor-2 }

      [ ;**=filedescriptor-3  ]

     [  ;*S=filedescriptor-4  ]

     [ { NDUMP }  =filedescriptor-5  ]
       { DUMP  }

     [ { NDECK     =  { ONLINE           } ]
       { {DECK}       { REMOTE(ii)       }
                      { filedescriptor-6 }
```

Format (cont.):

```
        ┌ NCOMDK                                    ┐
[ ;    <  ┌COMD  ┐    ┌ONLINE                  ┐    >]
        │ <      >  = < REMOTE(ii)              >   │
        └ └ACOMDK┘    └ filedescriptor-7        ┘   ┘ .

[ ;COPY=filedescriptor-8 [ ,filecode ] ]...
[ ;REPLACE  ]

[;RESEQ  [,(integer-1,integer-2) ]  ]

[ ;DWK  ]

  [ ; ┌NLSTIN┐          ]
      └LSTIN ┘

[ ;ALTNO  ]

[ ;ASCIIPRT   ]

[ ;MAP ]

[ ;DEBUG ]

 [ ; ┌NLSTOU┐  ]
     └LSTOU ┘

[ ;XREF ]

[ ;LNRSM ]

[ ;NCLIST ]

[ ;NWARN ]

[ ;NMESS ]

[ ;IDSLIST  ]

[ ;NRESET  ]

[ ;LL  ]

[ ;LIL  ]

[ ;HIL  ]

[ ;HL  ]
```

Format (cont.):

```
[ ;ANSI  ]

[ ;COBOL-7   ]

[ ;TP  ]

[ ;DSE  ]    See (1)

[ ;H*=filedescriptor-9 ]

[ ;CORE=integer-3 [,integer-4 ] ]

[ ;TIME=integer-5 ]

[ ;LIBRARY=filedescriptor-10 ] ...

[ ;fc=filedescriptor-11 ["altname"] ] ...

[ ;DSS= [ SCHEMA ] [ ,GO]

[ ;MAIN=name-1 ]

[ ;USE=string-2 ]

[ ;LOADOPTION= [ C74LNK  ] [ ,NOSETU ] ... ]

[ ;LINK=name-2 [ , [origin] [,option] ]
       [ ;ENTRY=name-3 ]
       [ ;OBJECT=filedescriptor-12 ] ...]
```

Syntax Rules:

1. Options may be entered in either uppercase or lowercase.

2. Options may be specified in any order, with the following exceptions:

   a. The LIBRARY option(s) must logically precede any MAIN, LINK, LOADOPTION, USE, ENTRY, or OBJECT options in order to be effective.

   b. The MAIN option must logically precede any LINK, ENTRY, or OBJECT options in order to be effective.

   c. The LINK, ENTRY, and OBJECT options are processed as encountered, so their placement affects the order of the content of the R* file that controls the loading process.

3. Options may follow the colon in the CRN command or may precede the source text in the source file, or both. Option lines in the source file must contain a "$" as the first nonnumeric character.

4. Multiple options on a single line must be separated by semicolons, but a semicolon need not follow the last option on a line, even if another option line follows.

5. Options are processed in the order encountered on the source file, followed by those appearing in the CRN command. In the case of contradictory options, the last option encountered is effective. This means that an option found in the source file may be overridden by a contradictory option in the CRN command except for options pertaining to the creation of the H* file. These options are mutually exclusive and may be used in the source file or in the CRN command, but not both. The H* options are : NOLOAD, NOGO, DSS, LIBRARY, MAIN, LINK, ENTRY, OBJECT.

6. In each set of options given in the CRN command general format, the default option is listed first.

7. A CRN command may span more than one line as long as the line is terminated by a semicolon.

Discussion

The COBOL-74 time sharing system provides the capability for compiling loading, and executing COBOL-74 programs from a terminal under the control of the time sharing executive. Additional information regarding CRN options may be obtained from the COBOL-74 Users Guide (DE02).

Purpose

The CRUN command initiates command file processing. Command file processing is a noninteractive mode within time sharing during which user inputs are obtained from a file instead of the terminal. The output from the CRUN may be directed to the user's terminal or collected on a file for later examination.

Format

```
CRUN {infile};[otfile];[option-1;option-2;...option-n

{infile}  ::= {filedesc}
[otfile]  ::= [filedesc]
      option-i  ::= nn...n Maximum processor time limit

                ::= arg-1,arg-2...arg-n Substitutable arguments

                ::= ARG/c Substitution-implying character

                ::= DEL/c Delete character

                ::= INCLUDE or EXCLUDE Output file user response
                    disposition

                ::= UPPER or LOWER Output file case
```

Discussion

Command file processing is initiated upon receipt of the CRUN command. CRUN parameter declarations and syntax conventions are nearly identical to DRUN--the command used to schedule a deferred session. The execution phase of such a session constitutes a command file application. It is essential to review the DRUN command as it contains instructions for input file preparation and describes the functional characteristics of command file processing.

When no parameters accompany the command, the output file description is optional, but when present must constitute the second parameter. When only an input file description is specified, or is followed by two consecutive semicolons, all keyboard output generated for the command file application is directed to the user's remote device upon completion of the application. Output file management and description variations are similar to DRUN conventions, with the exception that a temporary file is created for the user when (1) the named file does not already exist, and (2) the description consists solely of a file name. The current file may be declared for either the input or output description (both, if desired) and is denoted by an asterisk (*).

The input file for a CRUN application may optionally contain line numbers and/or first line parameter declarations. Parameters unique to deferred processing applications are ignored when encountered as such first line declarations. This relaxation permits the same input file to be utilized for either a command file or deferred processing application.

The presence of line numbers on the input file is determined by the occurrence of a digit (0-9) as the first nonblank character of the first line. When an input file without line numbers otherwise begins with a numeric, an initial line may be introduced containing only the characters, ##; e.g.,

```
##
100    DIMENSION DATE(2)
200    CALL DATIM(DATE,TIME)
300    PRINT  100,DATE,TIME
400    100 FORMAT("ODATE: ",2A4," - TIME:",F7.3)
500    STOP;END
END
```

The input file illustrated above is shown merely for the sake of example and would have limited utility for a CRUN application, since it assumes the user is in build mode on behalf of the FORTRAN system selection and expects the current file to be in an initialized state. These shortcomings, in addition to the ## line requirement, can be overcome by replacing the ## line with the command:

```
*FORT NEW
```

The response of FORT NEW results in establishing FORTRAN as the current system selection with an empty current file.

The CRUN subsystem can be entered from another subsystem by preceding any recognizable command with a quote (").

$*$ FUNCTIONS

Various special response lines are recognized by the TSS Executive when command file processing is in progress. Such lines are identified by a leading string of "$*$". $*$ response lines may appear anywhere in the input file; i.e., they are not restricted to recognition at system selection or build mode levels.

$*$BRK

The response line $*$BRK may appear anywhere in the input file for a command file or deferred processing application and simulates a break when the response line is processed. The break will not terminate the CRUN or DRUN application, but rather will behave as though the break key was depressed during a live session.

$*$COPY

$*$COPY causes the Time Sharing Executive to send (to the user's terminal) a copy of all input data read and all output produced by the commands. The function is deactivated by the directive, $*$COPY OFF.

$*$DELE

$*$DELE causes the Time Sharing Executive to refrain from writing output to the *CFP file during command file execution. Output deleted by this feature is irrevocably lost. The function is deactivated by the directive, $*$DELE OFF.

$*$FILE

$*$FILE resumes command file mode of processing.

$*$LBL

$*$LBL defines a label (associated with the following line) which may be referenced as a transfer point with the CPOS command. A maximum of eight labels may be defined, each of which must consist of one to nine alphanumeric characters, including the period and dash. Commentary information may be included on a $*$LBL line, separated from the label declaration by a colon or semicolon. A label referenced with the CPOS command must constitute the operation subfield (second parameter) of the command and be preceded by a dollar sign ($).

The following examples are thus functionally equivalent in all respects:

Example 1:

```
CMOD 0
$*$LBL REPEAT-IT
LIST
CMOD +1
CPOS LE5;$REPEAT-IT
```

Example 2:

```
CMOD 0
LIST
CMOD +1
CPOS LE5;-2
```

Two special labels, ..BREAK and ..ABORT, may optionally be defined with the $*$LBL function. Each time the break key is pressed during execution of a CRUN application, control is passed to the response line associated with the ..BREAK label, provided it has been defined. It is important to note that no correlation exists between the ..BREAK label and the $*$BRK function; i.e., $*$BRK invocations do not transfer control to the ..BREAK label. Similarly, the occurrence of any error in either a CRUN or DRUN application (which will result in abnormal termination of the application) causes control to be passed to the response line associated with the ..ABORT label. When termination occurs, or if a second error is encountered, control is either passed to the ..ABORT procedure of the previous level of processing (in the case of nested CRUNs), or the application is terminated. The reason code (1-16) for the abort is available to the user's ..ABORT procedure and may be materialized in the lower half of the Program Switch Word via the "ERROR" (or more simply "E") option of the CMOD command; e.g.,

```
$*$LBL ..ABORT : COME HERE IF ABORT OCCURS

CMOD E

$*$REM DIRECT OUTPUT TO *SRC IF ERROR CODE≥6

CPOS GE6;$SKIP

COUT *

$*$LBL SKIP
```

Error codes (decimal) and their associated meaning are as follows:

Code#	Meaning
1	COMMAND FILE NONEXISTENT
2	COMMAND FILE I/O ERROR
3	COMMAND FILE FORMAT ERROR
4	EXCESSIVE OUTPUT GENERATED
5	INVALID USE OF DRL T.CFIO
6	USER ERROR #nnn DETECTED BY TSS
7	PPT START OR LINE SWITCH ATTEMPTED
10	PROCESSOR TIME LIMIT EXCEEDED
11	INPUT LINE LENGTH TOO LONG
12	COMMAND FILE INPUT EXHAUSTED
13	TERMINATED BY DABT WHILE EXECUTING
14	ERROR DETECTED IN SSname PROCESSING
15	SYSTEM LEVEL SYNCHRONIZATION ERROR
16	$*$FUNCTION NOT PERMITTED

The error numbers referenced in code 6 text are listed in Appendix A. The SSname in code 14 represents the name of the subsystem in execution at the time the error was detected.

$*$MARK

When the $*$MARK response line is encountered during a command file application, the message text starting in character position 9 is issued to the user's remote device. The MARK function can appear anywhere on the input file and may be used to notify the user when specific stages of processing are reached. The function is ignored when encountered in a deferred session.

The following example utilizes the function to display the snumb of the last batch job submitted by the user:

```
##;EXC;(#S)
$*$MARK LAST JOB SUBMITTED WAS #1
```

$*$NULL

The $*$NULL response line is the equivalent of a line containing a null response (CR only).

$*$QUIT

The $*$QUIT response line may appear anywhere on the input file or may be given as an interactive response. Its purpose is to immediately discontinue command file processing. The output file is not produced, nor is *CFP removed from the AFT.

$*$REM

The $*$REM function provides a means of entering commentary notes on the input file. Since it is not included as a response line on *CFP, care must be exercised in calculating the relative offset for CPOS repositioning; i.e., all $*$REM lines must be ignored for such calculations.

$*$TALK

$*$TALK permits the user to become interactive with his command file application. Upon receipt of this command, the last buffer of output generated on *CFP (normally the last line requesting an input response) is issued to the user's remote device. From this time until receipt of a $*$FILE command, all dialog exchanged between user and computer is directed to both the remote device and *CFP. Normal command file processing is resumed when $*$FILE is issued as the response for any input request. The actual response for this request is obtained from the response line on *CFP following the $*$TALK response line.

An interesting application of $*$TALK is to obtain a printer listing of the log produced by a Time Sharing session. This is easily accomplished by initiating a CRUN of the following input file at the beginning of the session:

```
##*NULL;DEL/
$*$TALK
COUT TEMPF
JPRINT TEMPF:IDENT(M26KLC554,JOHNDOE,STATION G)
```

The printer listing is produced when the $*$FILE command is given.

$*$TRAP

A command file or deferred processing application is immediately aborted upon the occurrence of any error or exception condition detected by a subsystem. In some instances, this action may be undesirable; e.g., use of a certain command might knowingly cause what appears to be an error, but has no effect whatsoever on the application. The $*$TRAP ON (ON is optional) and $*TRAP OFF response lines provide a means of enabling and disabling this feature at will. All applications are originally initiated with the trap mode enabled; however, nested CRUNs are initiated with whatever mode is currently in effect.

When the trap mode is disabled, the user can determine the success or failure of the command executed with the previous response line by testing bit 13 of the program switch word with the CPOS command. If the previous command was successfully executed, bit 13 will be reset (off); otherwise it will be set (on). Thus, in the following example, the JRN command is not issued if the required file fails to become the current file.

```
##*NULL;(FILENAME?)
$*$TRAP OFF
OLD #1
CPOS 13;+2
JRN
```

Note that it is necessary to examine the bit immediately following the command in question; i.e., other response lines cannot intervene between the command and the CPOS used to test its outcome.

$*$USER

$*$USER is a special form of $*$TALK, allowing a single response line to be supplied by the user. As with $*$TALK, the last line requesting an input response is issued to the user's remote device; however, upon receipt of the response, command file processing is automatically resumed.

In some instances it may be desirable to have a specific input request issued to the user's remote device upon receipt of a $*$TALK, $*$USER or $*$QUIT response line. This may be accomplished by affixing an equal sign (=) to the $*$ function, followed by the desired prompt message. For example, the response line,

$*$USER=WHAT IS YOUR ACCOUNT NUMBER?

results in issuing the question, "WHAT IS YOUR ACCOUNT NUMBER?" to the user's remote device. The output file is not affected by this feature and will contain the true prompt.

Command File processing with VIP devices is permitted; however, the response lines $*$COPY, $*$USER, $*$TALK, $*$MARK, and $*$QUIT cannot be utilized.

CRUN Processing Pushdown

Nested CRUNs may occur in a command file or deferred processing application to a maximum depth of four levels. The effect of encountering a CRUN during such an application is to suspend (pushdown) its processing until the new requested command file application terminates, whereupon processing of the previous application is resumed (popped up). When an error occurs during nested command file processing, output accumulated for all levels is generated and the entire application terminated. Note that each level of nested processing requires a command file in the user's AFT. Such files are identified by names *CFP through *CFT and care must be exercised to avoid AFT removal during the application. NEWUSER processing preserves these files for the new user, thus allowing a command file application to change user-IDS at will. Both user-ID and password must follow a NEWUSER command on the CRUN input file. The following example illustrates a command file application to change user-ids with no intervening output issued:

*CRUN "NEWU\JDOE\JDPASSWD";*NULL

A break or disconnect (via DRL DRLDSC) may be used to prematurely terminate a command file application. Output generated for the application up to the point of interruption is directed to the requisite file(s), or discarded when destined for the user's remote device.

COMMAND LOADER INTERFACE

The Command Loader Subsystem may be utilized to initiate a command file application. This subsystem is invoked upon receipt of any unrecognized command. Such a "command" is construed to be a file description conforming to one of the following conventions:

- o catalog/filename
- o /filename
- o filename (implies CMDLIB/filename)

The mode (random vs. sequential) of the target file dictates the function to be performed; i.e., random infers loading the resident bound program and passing control to it, while sequential implies initiation of a command file application with the designated file used as input. For the latter case, optional parameter declarations may accompany any of the file description conventions, consistent with CRUN command constructs and syntax. Assume, for example, the following input file is resident in CMDLIB with general read permission under the name, SIEV. Its purpose is to support a local command called SIEVE which produces a list of the user's first-level catalog and file names containing a specified character pattern.

```
##*NULL;(PATTERN?)
"CATA";*
-D;4
"-PS:/#1/;*"
```

The application may be initiated by typing SIEVE (or SIEV), whereupon the query, "PATTERN?", is issued to obtain the desired character pattern. The pattern may alternatively accompany the SIEVE command, as follows:

*<u>SIEVE;;(FIL)</u>

Note that this represents the equivalent of:

*<u>CRUN CMDLIB/SIEV,R;;(FIL)</u>

When only substitutable arguments must accompany the file name, it is permissible to substitute a space for the double quotes and left parenthesis. Thus, the SIEVE command could equivalently be entered as follows:

*<u>SIEVE FIL</u>

The following example utilizes COUT to place a specified SYSOUT report (generated by a JRN job with JOUT disposition) in the current file:

```
##*NULL;(SNUMB?,ACTIVITY?,REPORT-CODE?,DISP?)
JOUT #1
ACTIVITY #2
PRINT #3
#4
COUT *
-D;5 F;*
-B;4 D;4
```

Assuming this input file is resident in the user's own catalog under the name "EXTRACT", an example application might be initiated as follows:

```
*/EXTRACT 6837T,1,74,RELEASE
```

The CMOD and CPOS commands, while not entirely restricted for use in command file applications, permit count-controlled repetition or conditional execution of a command sequence.

Purpose

The DABT command permits deferred sessions originated by the user to be aborted or inactive entries on the deferred queue file to be removed.

Format

DABT [nnnnD; nnnnD;...;nnnnD] [ALL] [REMO]

[nnnnD] ::= deferred job number

[ALL] ::= abort all scheduled jobs for this user

[REMO] ::= remove all of user's aborted or completed DRUN jobs from deferred control file

Discussion

The DABT command may be utilized to abort one or more deferred sessions which may not have terminated. An "ALL" request aborts all jobs scheduled by the user without regard for a specified starting date or time. Jobs which are scheduled or executing on behalf of the user are candidates for aborting.

The REMO parameter serves as a cleanup mechanism to remove previously aborted or completed deferred sessions from the deferred queue file.

If the parameters do not accompany the DABT command a request for "JOB ID?" will be issued to the terminal. A line ending in a semicolon results in a continuation request for more input.

The MASTER user has the capabilities using DABT ALL* to abort all deferred sessions or using DABT REMO* to clean out the entire deferred queue file.

Examples

1) DABT 1234D

2) DABT 1234D; 2345D

3) DABT ALL

4) DABT REMO

Purpose

The dataBASIC command envokes the dataBASIC subsystem for entering, compiling and running dataBASIC programs.

Format

DATA[BASIC]

Discussion

Refer to the dataBASIC System Language Manual for details of the language.

Purpose

The DECODE command is used to obtain the original version of an encrypted file and is the complementary action to retrieve the contents of a file built using CODE.

Format

```
DECO[DE] [<infile>[;<otfile>]]

<infile>  ::= <filedescr>
<otfile>  ::= <filedescr>
```

Discussion

Input and output file descriptions are required as for CODE and are treated in a similar manner. The output file may be * to designate the current file. Upon receipt of the DECODE command, the user is asked to enter the encryption key. This is the same key originally used with the CODE command to encrypt the file. Key verification is not required with DECODE.

Example

```
1)   DECODE   SAFEFILE

2)   DECO     SECRET;NONSECR
```

Purpose

The DELETE command deletes a specified line or lines from the current file.

Format

```
[ DELE[TE] [ <line-ref>[,<line-ref>]...]

<line-ref>   ::= <line>|<line-range>
<line-range>::= <begin-line>-<end-line>
              |   <begin-line>-
              |   -<end-line>
<begin-line>::= <line>
<end-line>   ::= <line>
<line>       ::= a 1- to 8-digit decimal number
```

Discussion

The lines in the current file must have line numbers beginning in column 1 and the numbers must be steadily increasing.

The option -<end-line> is only accepted as the first <line-range> (Example 3).

The option <begin-line>- is only accepted as the last <line-range> (Example 5).

The lines and line ranges must be given in ascending order.

Examples

1) DELETE 10
   (deletes line 10)

2) DELETE -50
   (deletes the lines from the beginning of the current file through line 50)

3) DELE -15,200,220-275,302,309-410
   (deletes the lines from the beginning of the current file through line 15, line 200, lines 250 through 275, line 302, and lines 309 through 410).

4) DELE 460-
   (deletes lines 460 through the end of the current file)

5) DELETE 13,25,70-90,999-
   (deletes line 13, line 25, lines 70 through 90, and lines 999 through the end of the current file)

Purpose

The DISPLAY command is the CONVERT command used to produce formatted lists of file contents.

Format

```
DISP[LAY][infile][:options]

[infile]     ::= [filedescr]
[:options]   see below
```

Description

Media Code Options

The output record format options specify the physical format of the output record. The default option for the CONVERT command is "ASCII". A list of the options and their meanings is as follows:

BCD - variable length BCD - media code 0

COMDK - BCD compressed deck card image (COMDK) - media code 1

CARD - BCD 14-word card image - media code 2

PRINT - BCD variable-length print line image - media code 3

OLDASC - obsolete TSS ASCII - media code 5

ASCII - standard system format ASCII - media code 6

APRINT - ASCII print line image - media code 7

ACARD - ASCII card image - media code 10

SAME - a record output media code is the same as its input media code

Line Number Options

Line numbers can exist with COMDK, CARD, ACARD, OLDASC, and ASCII records. All BCD, PRINT, and APRINT records cannot possess line numbers. The line number for an ASCII or OLDASC record consists of 1 to 8 numeric characters. These numeric characters must be among the first eight characters in a line. A line number is defined to include any leading blanks. A line number is terminated by a nonnumeric character, including blank. If the "#" character terminates a line number and if it is one of the first eight characters of a line, it is considered to be a delimiter. It is treated as neither part of the line number nor part of the text. The line number for COMDK, CARD, and ACARD records is defined to be all the trailing digits in columns 73-80. This field may begin with nonnumerics; these also are considered neither part of the line number nor part of the text.

The line number options may specify:

1. Whether line numbers are to appear in the output text.

2. The actual line number values.

The default line number option is "ASIS". A description of each of the options follows:

ASIS Line numbers are assumed not be present in the input file. Text, including leading/trailing numeric characters and "#"'s are left as is.

STRIP Strip line numbers from the input text before reformatting and writing the output text. Input COMDK, CARD, and ACARD records are truncated at column 72. Line numbers on ASCII and OLDASC records, when present, are discarded and the first character following the line number is treated as the first character of the line.

MOVE Move line numbers. The input records have the line numbers detached from the text string, either from the front (ASCII or OLDASC) from columns 73-80 (COMDK, CARD, or ACARD). The output records have the line numbers reattached to the text string, either at the front (ASCII or OLDASC) or in columns 73-80 (COMDK, CARD, or ACARD). If the output records are BCD, PRINT, or APRINT, the line numbers are not reattached and the M option acts similar to the S option.

I(i,j) Insert line numbers beginning with line number j and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. The input file is assumed not to be line-numbered. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the I option is ignored.

R(i,j) Resequence line numbers. Strip any existing line numbers from the input text and insert new line numbers in the output text, beginning with j and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the R option behaves as the S option.

N(ch) Implies the M option and specifies that the normal tab character (the colon) and tab settings (8, 16, 32, 73) have been employed in building the input file(s). The (ch) argument may be used to define a character which replaces the colon as the tab character.

LABEL (abcde(i-j)fghi(i-j)---) If the output records are COMDK, CARD, or ACARD, then a label is placed left-justified in columns 73-77. The label is specified as 1 to 5 nonblank characters. The fields "abcde" and "fghij" represent the labels. The label is placed on only those lines with line numbers between i and j inclusive. Up to 10 distinct labels may be given. If more than one label is given though, the (i-j) specifications may not overlap.

The LABEL option is meaningful only if line numbers are attached to output records. Therefore, the label option is completely ignored unless it is accompanied by either the insert, resequence, or move option.

For the I and R options, output line numbers for ASCII and OLDASC records will have at least the number of digits specified for i in I(i,j) or R(i,j). Thus R(0010,10) will result in line numbers 0010, 0020, 0030,---.

Input records are assumed to have line numbers when the STRIP, MOVE, and RESEQUENCE options are specified. Otherwise, line numbers are assumed to be absent and leading numerics in ASCII format are treated as real text. When line numbers are assumed present, tabbing and columnizing are performed relative to the start of the real text.

The user must be careful not to alter the line number values of a BASIC file.

Character Manipulation Options

A description of each of the character manipulation options follows.

TAB(ch,i,j---;ch,i,j---;----) Expand tab characters into blanks. Use "ch" as a tab character with settings i,j,k,etc. Usually, any occurrence of the tab character in the input file(s) results in the replacement of the tab character with a string of blanks up to the next tab setting. However, if a tab character is encountered beyond the last tab setting specified for that tab character, it is treated as a normal nontab character.

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the tab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

UNTAB(ch,i,j---;ch,i,j---;----) Insert tab characters, replacing blanks. Use "ch" as a tab character with settings i, j, k, etc. Any occurrence of a string of blanks terminating on an "untab" tab stop is replaced by the character "ch".

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the untab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

LOWER	Convert all alphabetic characters to lowercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.
UPPER	Convert all alphabetic characters to uppercase. This option is meaningful only if the output records are ASCII, OLDSAC, or APRINT.
BEGIN(ch)	Begin a new line (record) immediately after the character "ch". The character "ch" is treated as a delimiter and not part of the text. It is not placed in the output text. When the "ch" character is located at the beginning or end of a line, it is simply deleted. Strings of the "ch" characters are treated as a single "ch" character.
COLUMN(i-j)	Delete all of the characters in a line except those which are located within columns i through j inclusively. The options BEGIN and TAB are both completed before COLUMNS takes effect. If a record does not extend through column j prior to the COLUMNS option execution, it is blank filled to column j. Thus, when the COLUMNS options is in effect, the length of all generated output records is j-i+1 characters.
SQUEEZE	Replace any string of two or more blanks by a single blank. The options BEGIN, TAB, COLUMNS, and UNTAB are all performed before SQUEEZE is executed.
TRAIL	Delete all trailing blanks on a line. The TRAIL option is performed immediately after the SQUEEZE option.

A number of options affect the length of an output text line. It is important that the user understand the order in which these options are performed. The order (from first to last) in which the options are executed is:

BEGIN
TAB
COLUMNS
UNTAB
SQUEEZE
TRAIL

Miscellaneous options

VERIFY	If the VERIFY option is in effect when CONVERTY completes the processing of an input file, then CONVERT gives a brief summary of the number of records obtained from the file. This summary gives, for each media code, the number of records which had that media code.
IGNORE	Ignore all embedded $$ control lines. Treat them as text.

DISCARD Discard all nontext records. Nontext records are those records which media code is not one recognized and interpreted by CONVERT. The JRN, JPRINT, JPUNCH, APRINT, and DISPLAY commands require that nontext records be discarded. The CONVERT command normally does not require that nontext records be discarded. When nontext records are encountered during the execution of the CONVERT command, they are written to the output file, but no reformatting or media conversion is performed.

TIME When the TIME option is invoked, the date and time of day are printed at the user's terminal.

DEFAULT The DEFAULT option is used to nullify all options which the user has specified either on the command line or embedded $$ control lines. The default option has no affect on any of the "specialized" options. Because of the nature of the DEFAULT option, it is meaningless for it to be located in the options field of the command line. Therefore, if the DEFAULT option is encountered in the options field, an error message is issued. The same reasoning applies to the placement of the DEFAULT option anywhere other than the beginning of a $$ control line.

File Processing Options

SELECT (file) The SELECT option is analogous to the $ SELECTA card. The select option allows an input file to specify other input files. Upon encountering the SELECT option, the selected file is obtained and is used in place of the $$ control line. Nesting of selects is permitted up to 17 levels. The SELECT option is meaningful and valid only on a $$ control line. Only one SELECT option may be specified on control line.

INCLUDE If the INCLUDE option is in effect, CONVERT, upon encountering the SELECT option, uses the selected file as an input file.

EXCLUDE If the EXCLUDE option is in effect, CONVERT ignores the SELECT option.

The purpose of the INCLUDE and EXCLUDE options is to allow the user to control the performance of the select options while not forcing him to disregard:

1. Other options on the same $$ control line.

2. All $$ control lines.

The INCLUDE option is the default option for the JRN command. The EXCLUDE option is the default option for the JPRINT, JPUNCH, APRINT, DISPLAY, and CONVERT commands.

Discussion

```
DISPLAY FILEA :M,T(%,10,20;\,30,40,50)
```

The contents of FILEA are printed at the user's terminal. Line numbers are in columns 1 through 8 and text begins in column 9. Tabs are expanded. The tab characters are % and \. The settings for % are 10 and 20. The settings for \ are 30, 40, and 50.

Purpose

The DMIV command invokes the DM-IV Query and Reporting Processor.

Format

DMIV

Discussion

Refer to DM-IV Query and Reporting Processor (QRP) Basic User's Guide for further information.

Purpose

The DONE command allows a user to leave a system or subsystem so that another can be entered.

Format

DONE

Discussion

To enter a different system, the user must first leave control of the current system.

Purpose

The DRUN command initiates a deferred processing job optionally at a later time and date.

Format

```
DRUN {infile};{otfile};[option-1;option-2;...option-n]

{infile}     ::= {filedescr}
{otfile}     ::= {filedescr}
option-i     ::= nn...n Maximum processor time limit
             ::= arg-1,arg-2,...arg-n Substitutable arguments
             ::= ARG/c Substitution-implying character
             ::= DEL/c Delete character
             ::= INCLUDE or EXCLUDE Output file user response disposition
             ::= UPPER or LOWER Output file case
```

Discussion

The term "deferred processing" implies a planned time sharing session scheduled by a user to be independently initiated at some given date and time. Since the user does not actively participate in a dialog exchange with the computer during the session, the user must anticipate responses and provide an input file containing them in the order to be presented. An output file must also be provided to collect the exchanged dialog. A listing of this file produced after the deferred session terminates will appear notably similar to the log of an online user.

Upon receipt of the DRUN command, the user's input and output file descriptions are obtained, in addition to any optional parameter declarations. The job is then assigned a unique identifier (nnnnD) and recorded in a special deferred queue file with a status indicating that it is scheduled for initiation. After establishing the request for deferred processing, the user may continue his online session or disconnect if desired. When the job becomes eligible for initiation, it is logged on with the user-id of its originator and the status of the deferred queue entry is changed to indicate the session is in progress. When termination occurs, the status is again updated to reflect a normal or abnormal ending and the deferred session is concluded.

If no parameters accompany the command when it is issued, the user is asked "FILE NAMES?" and must, at this time, enter all information necessary to schedule the deferred session. Input and output file descriptions are required and must constitute the first two parameter declarations. Optional parameters may follow the output file description in any order desired.

The input and output file descriptions declared for a deferred session must conform to the conventional TSS format. These files must be permanent and up to three levels of subcatalogs may be specified for each. This limit is imposed by the space available on the deferred queue file and, when substitutable arguments are declared, is further reduced to a combined total of four subcatalog levels. All passwords necessary to allocate either file must accompany its description. A line number interval may optionally be declared for the input file and either description may include an alternate name and/or permissions.

Continuation input is requested for a file description (or the next ## line of the input file obtained) when the input line is prematurely terminated with a slash, dollar sign, comma, left parenthesis or the leading quote of an alternate name declaration. The semicolon, used to separate parameters from one another, always implies continuation when it occurs as the last character of the line.

Preparation of the input file is essential prior to scheduling a deferred session. This file may optionally have line numbers, the presence of which is established by the occurrence of a digit (0-9) as the first nonblank character of the first line. Line numbers serve no functional purpose for deferred processing other than allowing the user to specify only a segment (line-number interval) of the file to be processed. If present, they are sequence-checked and otherwise discarded. Note that as with JRN batch job submittal format, the pound sign (#) may be used to separate the line number from the text when the latter begins with a numeric character. Double pound signs (##) are required when the first textual character is itself a #.

DRUN parameters may optionally be provided on the first line of the input file (or input file segment), consistent with the following syntax:

```
##filedescr-out;option-1;option-2;...;option-n
```

The output file description is required and must constitute the first parameter of the line. If not provided, it must be declared explicitly null, as follows:

```
##;option-1;option-2;...;option-n
```

Parameter declarations supplied with the DRUN command override similar declarations on the ## line. Thus, when an output file description is included in both the DRUN command line and the ## line of the input file, the latter declaration is ignored.

A limited number of user responses required for a deferred session may be substituted for the input file description on the DRUN command line. All such responses must be enclosed in quotes and separated from one another by reverse slants (\) or ampersands (@).

The quoted string may be prematurely terminated anywhere, causing continuation lines to be requested until the occurrence of the end quote. Maximum string length is not easily predictable and is a function of the presence of substitutable arguments, the number of subcatalogs qualifying the output file, and the number of individual responses occurring in the string. In general, two or three lines can usually be accommodated. Note that two consecutive line delimiters or a line delimiter immediately preceding the terminating quote represents the equivalent of a null response. A quote, reverse slant or ampersand may be used as text in the character string by preceding it with an ESC (escape) character.

The output file for a deferred session is not utilized until the session terminates, whereupon its allocation is attempted in accordance with the file description declared when the session was scheduled. If the file is found to be nonexistent, it is created for the user, provided (1) an alternate name was not specified, or (2) requested permissions, if present, include only a subset of R/W/A/E. Permissions and/or a password may optionally be attached to the file when it is created by including them in the description. A special output file declaration, *NULL, may be provided if the user does not want to save his output.

The output file name declaration *ID may optionally be specified with the DRUN command, causing the job identifier (nnnnD) to be used for the file name. This declaration is also applicable for CRUN and CPOS, provided the command is executed in a deferred processing environment. *ID may be qualified by subcatalogs and permissions with which to create the file may accompany the description.

Example: *<u>DRUN INFIL;/CAT1/*ID</u>

Optional parameters that may accompany a DRUN request include the following:

- Earliest Session Initiation Date

  Form: YY/MM/DD, YY-MM-DD, MM/DD/YY or MM-DD-YY

  This declaration indicates the earliest date on which the deferred session can be initiated. YY represents the last two digits of the year, MM is the month and DD is the day. MM or DD may consist of either one or two digits. If not specified, the current date is assumed.

- Earliest Session Initiation Time

  Form: HH:MM or HH.TTT

  This declaration indicates the earliest time of day on which the deferred session can be initiated. HH represents the hour (0 HH 24) and may consist of one or two digits, while MM indicates the minute within the hour and must constitute two digits. TTT is a 1-3 digit fractional hour specification. If not declared, the session will either be initiated at the earliest possible time on the target date or at a preferred time on that date optionally specified by the site.

o Maximum Processor Time Limit

Form: nn...n

This decla ation, consisting of an integer representing seconds, limits the total processor time permitted for the deferred session. The site itself can impose such a limit for all deferred sessions, overriding the user's declaration if the latter is larger. If neither site nor user imposes this limit, the deferred session is allowed to run for an unlimited period of time.

o Substitutable Arguments

Form: (arg-1,arg-2,...,arg-n) or (arg-1;arg-2;...;arg-n)

Substitutable arguments provide a means for having the character string implied by the i-th argument declaration substituted for any occurrence of the characters #i appearing in the input file (the input file itself is not modified). Up to eight arguments may be specified, each consisting of 1-12 characters. Permissible characters include alphanumerics, the period, dash, colon, slash, dollar sign, pound sign and asterisk. Blanks included in a character string are retained and all alphabetics are forced uppercase.

In some instances it may be necessary to preserve the case of alphabetics and/or utilize characters not otherwise permitted for an argument. For such requirements the argument may be enclosed by quotation marks and constitute up to ten characters. An argument declared in this manner is unrestricted with respect to content (it may contain any ASCII characters) and is substituted exactly as it appears.

Example: DRUN INFIL;OTFIL;("/CAT/FIL,R","(10,100)")

Individual substitutable arguments declared on the DRUN command line normally override the corresponding argument declaration on the line of the input file; i.e., the latter declaration is ignored. When override is not desired, the specific command line argument may be declared null by entering two consecutive commas. Note that the number of arguments specified on the command line may differ from the number specified on the ## line of the input file. Missing arguments of the shorter list are treated as if they were explicitly declared null. The number of arguments substituted corresponds to the larger list.

When a question mark (?) occurs anywhere in a nonquoted substitutable argument, the entire argument string is issued to the user, representing a request to enter the actual value of that argument. Thus, the argument list, (YFOR,FILENAME?,LIST-OR-RUN?), declared for a DRUN results in issuing the questions, "FILENAME?" and "LIST-OR-RUN?" to the user for obtaining the actual values of arguments #2 and #3. A quoted string (as described above) may be entered if desired.

Certain special arguments representing specific information may be declared, as given in the following table:

Arg	Implies Substitution of	Format
#DATE	Current date	YY/MM/DD
#TIME	Current time of day	TT.TTT or T.TTT
#USERID	Logon user-id	XXX...X
#ACCOUNT	User account number	XXX...X
#CHANNEL	Channel number	NNNN
#SNUMB	Last snumb # generated	NNNNT

The character string which is substituted for such an argument declaration corresponds to the value of that argument when the DRUN command is issued. Thus, for example, the argument #CHANNEL equates to the channel number associated with the user who scheduled the deferred session. The special arguments may be abbreviated by the first character of their name. For example, #D is equivalent to #DATE. The following input file illustrates use of the #ACCOUNT argument to establish the same account number for the deferred session as was in use when the session was scheduled:

```
##OTFILE;(#A)
NEWU #1
JRN SAVEFILE
BYE
```

A series of 1-6 pound signs (##...#) may be declared for any substitutable argument and implies the substitution of a corresponding number of digits obtained by converting the lower half of the user's Program Switch Word (PSW) to decimal. When fewer than six pound signs are specified, leading digits of the converted PSW are discarded; i.e., the value substituted consists of the least significant digits. Leading zeros are included, when necessary.

The lower half of the PSW for a deferred session is initially set to the corresponding PSW value of the user who scheduled the session.

- o Substitution-Implying Character

  Form: ARG/c

  The characters #i are normally used to request substitution of the i-th argument. The ARG/c declaration permits the user to specify any character (c) other than # to denote argument substitution. This is necessary, for example, when the pound sign occurs as text in the input file.

- o Delete Character

  Form: DEL/c

  This declaration requests deletion of all occurrences of the specified character (c) appearing in the input file. Its use is primarily intended for preparing null response lines. Such a line might consist solely of the declared character. Note that an implied (default) delete character is not provided.

o Output File User Response Disposition

Form: INCLUDE or EXCLUDE

The output file produced for a deferred session normally includes the user's responses as they appear on the input file; i.e., the INCLUDE option is implied. The EXCLUDE option may be declared when the user wishes to eliminate all lines containing responses on the output file. Abbreviations INC or EXC are permitted.

o Output File Case

Form: UPPER or LOWER

Alphabetic text generated on the output file may be forced to uppercase or lowercase by the use of the appropriate option. If neither option is exercised, the case of alphabetic text is preserved. Abbreviations U and L are permitted.

o Deferred Session Restart

Form: RESTART

A system interruption occurring while a deferred session is in progress normally results in marking the session aborted when Time Sharing is restarted. The RESTART option, which may be abbreviated RES, requests that the session be reinitiated from the beginning if an interruption occurs.

Deferred sessions originated by a user which are scheduled to run at some later date and/or time may be rescheduled to run as soon as possible by utilizing the DRUN Command with the following syntax:

DRUN #job-id-1;job-id-2;...;job-id-n

Alternatively, the user may request all of his sessions scheduled for the current date which did not specify a specific start time to be initiated by issuing the following directive:

DRUN #ALL

Job identifiers declared with the DRUN, DSTS or DABT commands must belong to the requesting user. When the identifiers do not accompany the DSTS or DABT command on the same line, the query, "JOB ID?" is issued, at which time the identifier(s) must be declared. Continuation input is requested when the input line ends with a semicolon.

Functional Summary

A deferred session is logically divided into four stages of processing--scheduling, initiation, execution and termination. Scheduling occurs upon receipt of the DRUN command and consists of logging the request on the deferred queue file. The queue file, cataloged as specified in the TSS communication region definitions, serves as a collection medium for all information necessary to later initiate the deferred session. Such information includes the job identifier (nnnnD) assigned for the session, user-ID, starting date/time, processor time limit, input and output file descriptions and all substitutable arguments declared.

When a job is logged on the queue file, the next candidate for initiation is determined by the DRUN Subsystem and the TSS Executive is notified of its job identifier, user-id and earliest starting date/time. Note that due to TSS load conditions, it may not be possible to initiate the session at the exact prescribed time; however, when conditions permit, a UST (User Status Table) is developed for the session and the DRUN Subsystem is invoked with an indication that the deferred session is to begin. At this time, all information relevant to the session is retrieved from the queue file and a "command file" is prepared as for the CRUN command.

Preparation of the command file involves creating a random temporary file named *CFP and copying the input file to it, one line per hardware sector. Initial parameter lines are ignored, since they were processed when the session was scheduled. When requested, argument substitution and removal of DEL character occurrences take place during the copy; however, this is the extent of input file content analysis and no guarantee is made regarding legitimate command constructs or syntax. An initial and final sector is generated on *CFP, bordering the sectors developed from lines of the input file. The initial sector contains the user-id of the session originator and is utilized by the logon subsystem. The final sector contains a COUT command, accompanied with all information necessary to produce the output file when the session terminates.

After marking the status of the deferred queue entry to indicate the session is executing and notifying the TSS Executive of the next candidate for initiation, command file mode is enabled and control passed to the logon subsystem for user validation. During command file processing, the TSS Executive obtains each required user response by reading the appropriate *CFP sector and directs all generated output to *CFP starting at the sector immediately following the generated COUT command.

Normal termination occurs when a BYE or the generated COUT command is encountered at system selection level or while in build mode. In either case, the DRUN Subsystem is invoked to produce the output file and change the queue file status to indicate the deferred session has terminated. The output file description is obtained from the *CFP sector containing the generated COUT command, after which the file is created (if necessary) and allocated. If the EXCLUDE option was declared, the output segment only of *CFP is read, formatted and written to the user's output file. Otherwise, input and output segments are collated to produce the file. Collation is made possible by referencing the sector number of each user response in the appropriate sector of the *CFP output segment.

Abnormal termination can occur for a variety of error conditions detected by either the TSS Executive or the DRUN Subsystem. When possible, the output file for the aborted session is produced and in all cases the reason message for the abort is saved on the queue file for DSTS inquiry purposes.

Examples

1) The following example illustrates the input file for a deferred session that results in spawning a CARDIN job (user responses are underlined):

```
*AUTOX
*010 ##OTFILE
*020 JRN SAVEFILE
*030
*SAVE INFILE
*DRUN  INFILE
 DEFERRED ID 6772D
```

Line 010: Since this is the first line of the file, the ## identifies it as containing DRUN parameters. In this case, the only parameter given is the name of the output file.

Line 020: The JRN command causes the job images on file SAVEFILE to be sent to the batch environment for processing. File SAVEFILE must have all questions answered as first line default answers (e.g.,$IDENT?). The logon user-ID and password are implicitly provided by the DRUN process and do not have to be included in the file.

A listing of file "OTFILE" produced when the deferred session terminates might appear as follows:

```
HIS TIMESHARING ON 04/25/77 AT 11.583 DEFERRED #6772D TS1

USER ID-JOHNDOE

*JRN SAVEFILE
  SNUMB # 2367T
*BYE
**COST:  $  0.22 TO DATE:  $   316.53= 32%
**ON AT 11.583 - OFF AT 11.589 ON 04/25/77
```

2) The deferred session illustrated in previous example could be requested as follows:

```
DRUN"##OTFILE\JRN SAVEFILE\BYE"
```

or

```
DRUN "JRN SAVEFILE\BYE";OTFILE
```

3) The following DRUN reschedules itself to be rerun.

```
##*NULL;(FIRST-DAY?,LAST-DAY?,MONTH?,
##YEAR?,##)
CPOS NEO;+4
CMOD #1;+1
DRUN INFILE;OTFILE;#4/#3/#1;22:00;(#1,#2,#3,#4)
BYE
JRN SAVEFILE
CPOS GT#2;+3
CMOD +1
DRUN INFILE;OTFILE;#4/#3/#5;22:00;(#1,#2,#3,#4)
BYE
```

The input file (named INFILE) for the deferred session illustrated in the above example results in initiating a batch job, using the JRN command, on a specified day at 10:00 P.M. and rescheduling itself to be initiated the next day at the same time, continuing until (and including) a specified termination day. The example assumes the lower half of the PSW is initially zero and uses substitutable arguments to request starting date, ending date, month and year for the deferred sessions. Note that an immediate session is initiated to schedule the first requisite DRUN.

Purpose

The DSTS command requests the status of deferred job originated by the user.

Format

DSTS [nnnnD-1;nnnnD-2;...;nnnnD-n][ALL]

nnnnD-i::= deferred job number

[ALL] ::= all of user's deferred jobs

Description

The status message returned in response to the user query is one of the following:

nnnnD - SCHEDULED TO RUN yymmdd AT tt.ttt

nnnnD - RESCHEDULED TO RUN yymmdd AT tt.ttt

nnnnd - EXECUTING

nnnnd - TERMINATING

nnnnd - ABORTED BY DABT yymmdd AT tt.ttt

nnnnD - TERMINATED NORMALLY yymmdd AT tt.ttt

nnnnD - ABORTED yymmdd AT tt.ttt FOR REASON: reason text

nnnnD - ABORTED DUE TO SYSTEM INTERRUPTION

The acronym ASAP (As Soon As Possible) is substituted for date and time in the status message for a scheduled or rescheduled when it is overdue for initiation. The MASTER user may obtain the status of all deferred jobs by issuing a DSTS ALL*.

Example

```
*DSTS 1234D
1234D - EXECUTING
```

Purpose

The EDITOR command calls the Text EDITOR subsystem into use.

Format

EDIT[OR]

Discussion

The Text Editor subsystem is a means by which lines of text may be entered and edited in the current file. A line of text is any line with or without a line number. Lines of data or program statements can be considered as text for editing purposes.

The EDITOR can be entered from another subsystem by preceding any recognizable command with a hyphen (-). When a hyphen followed by a recognizable command is issued from another subsystem, the EDITOR does not determine if there is data on the *SRC (current file). If the entry is made without a *SRC, the EDITOR creates one and returns to the user with a hyphen, or returns to the calling subsystem level, as appropriate.

Refer to a later section for details of the Text Editor subsystem.

Example

*EDIT

Purpose

The ERASE command overwrites (erases) a specified file(s) with zeros, but does not release the file(s) from the file system.

Format

```
ERAS[E][&lt;filedescr-1;filedescr-2;...;filedescr-n]
```

Discussion

The ERASE command provides a user with the ability to purge the contents of a file when the contents are no longer needed or are of a sensitive nature.

Purpose

The FDUMP command calls the FDUMP subsystem into use.

Format

FDUM[P] [{*|<filedesc>|<filedesc><MODE>}]

```
<MODE> ::= ;R  access file as random if possible
           ;L  access file as linked if possible
```

Discussion

The FDUMP subsystem provides for file inspection and maintenance. It allows the user to look at the actual binary contents of a file. For system standard files, this includes being able to look at Block Control words, Record Control words and end-of-file marks. These are part of a file, but are not considered part of the data content of the file. They do not, for instance, show up as part of a LISTing or as part of a READ.

The user specifies the words that are to be looked at (snapped) and they are printed in the octal representation of the binary words. The user can modify (patch) any word by specifying the new contents of the word in octal. Since FDUMP uses octal representation for its functions, it is mainly used as a debugging tool.

FDUMP begins its file analysis by asking

BLOCK TO BE READ -

The permissible responses are:

carriage return - return to the FILE NAME? level.

<u>n</u> - the block specified by the block serial number <u>n</u> will be read into an internal buffer. The <u>C</u>opy function (see following function description) requires a dummy response of 1.

The response to "BLOCK TO BE READ" must take into account the mode of the file being processed. If the file is linked, <u>N</u> begins with 1 and represents 320 word blocks of the file. If the file is random, <u>N</u> may range from 0 to the last physical sector of the file. For random files the units of n are 64 words (e.g., to dump the second 320 word block of data on a random file would require N=5).

If the block serial number is outside the limits of the file, the error message BSN OUTSIDE FILE LIMITS is given, and BLOCK TO BE READ is repeated. If the block serial number is within the current file size but the implied block was not written on the file, it is read but it contains garbage data not pertaining to that file.

The third level (and final) question is:

FUNCTION?- (this question is repeated upon return from any of the FDUMP func ions.)

The permissible responses are:

carriage return - return to BLOCK TO BE READ.

Sloc - Snap the specified (octal) location.

Sloc-loc - Snap the field specified by (octal) location-location (from-to).

Sloc,n - Snap n (octal) words starting with the specified (octal) location.

Ploc data - Patch the specified (octal) location with the specified (octal) data. The data entry may be a multiple entry. For example:

```
Ploc datal, data22 data3
   data1 goes to loc
   data2 to loc+1
   data3 to loc+2
```

W - Write the corrected block back into the permanent file.

C filedesc - Copy the complete file onto another file specified by filedescr.

D- Done. Return to previous level of processing.

F loc - (Find data pattern)

The latter function (F) may be used to find the location(s) of one or more occurrences of a specified data pattern (D1) in a block of data with the search commencing at any designated location (loc). An optional mask (D2) may be provided to enable comparisons only on selected bit positions. If provided, bit positions of D2 which contain a 1 will cause the corresponding bit positions of D1 to be ignored during the search. If the mask is not specified, comparisons will be based on a full 36-bit word.

Permissible forms of the F function are given below.

Form	Meaning
F A1,D1	Find the first occurrence of D1, starting at location A1.
F A1,D1;n	Find the first n occurrences of D1, starting at location A1.
F A1,D1;*	Find all occurrences of D1, starting at location A1.
F A1,D1,D2	Find the first occurrence of D1 masked by D2, starting at location A1.
F A1,D1,D2;n	Find the first n occurrences of D1 masked by D2, starting at location A1.
F A1,D1,D2;*	Find all occurrences of D1 masked by D2, starting at location A1.

If the search is successful, each address at which the data pattern, D1, was found is displayed. In addition, if a mask has been specified, the data content at each address is displayed.

If the search is unsuccessful, the message "PATTERN NOT FOUND" is issued.

Examples:

F0,56060062056	(Find ASCII ".02." starting at octal location 0 (zero) in the block of data being scanned)
F100,2000,777777000777	(Find all DRL OP codes starting at octal location 100 in the block of data being scanned)

Once the user has become familiar with the conversational, or question/response sequence, form of FDUMP, a short form of function specification which effectively eliminates questions normally asked can be used. Multiple responses can be supplied on a single line, separated from one another by a space character. For example:

```
*FDUMP * 1 S50,100 DONE
```

NOTE: n consecutive spaces represent the equivalent of n-1 null responses.

The FDUMP copy is a physical copy; that is, it does not stop at logical end-of-file but continues to the file length defined in the file system as current size.

If the copy file is smaller than the size defined for the file to be copied, FDUMP grows the copy file to the necessary size.

Note that filedescr, specifying the copy file, may be simply a file name or may be a catalog/file string, but must not have the same filename even if duplicate filename is under a separate catalog.

At completion of the copy, the user is returned to the FILE NAME? level.

The FDUMP subsystem may return one of the following error messages during processing.

1. When the named file cannot be accessed, FDUMP replies

   CANNOT ACCESS FILE filename

   and returns control to the calling level.

2. When the block serial number given is either zero or is a number larger than the possible number of blocks in the file, the error message is:

   BSN OUTSIDE FILE LIMITS

   BLOCK TO BE READ is then repeated.

   For linked files, block size is assumed to be 320 words; the first block serial number is 1. Random files are positioned by multiples of 64 words, beginning with block 0. However, they are read in blocks of 320. Therefore, one read makes available five contiguous blocks of 64 words.

3. When the system receives a bad hardware status, FDUMP replies:

   <51>FILE filename -- I/O STATUS xx

   FUNCTION?

   A partial block may have been read and may be correctable by use of the S, P, and W functions. If none of the block appears to have been read, a carriage return answer repeats the BLOCK TO BE READ question. Then the block serial number that was specified can be verified.

4. When parameters are incorrect in form for the S, P, or W functions, FDUMP will reply

   INVALID INPUT-RETYPE

Examples

```
1)   *FDUMP           (prompts for the file name)

2)   *FDUMP DATA75

3)   *FDUMP *

4)   *FDUM E

       FILE IS RANDOM

       BLOCK TO BE READ? 0
       FUNCTION ? S0-47
       000000  000001000000    000000000000    252431634651    000240000002

       000004  000000000000    000000000000    000000000000    000000000000

       ? D
       *FDUM E 1 S0-17
       FILE IS RANDOM

       000000  000000000000    000242000022    000000000000    000000000000

       000004  000000000000    000000000000    000000000000    000000000000

       ? D
```

Purpose

The FORM command allows the Time Sharing Executive to transmit a form feed character after each page request.

Format

FORM

Discussion

FORM operation of a keyboard/display device causes the screen to be cleared after each page request. NFORM operation overrides the preceding page specification.

The FORM command is not applicable for a non-VIP keyboard display terminal under control of the PAGE command.

Purpose

The FRN command compiles, loads, and/or executes a FORTRANY time sharing program.

Format

```
FRN[-<time>][ <source-file>][=[<memory-image>]

   [;<object-file>][ (<options>)[<user-library>]]]

   [#<run-time-file>]

<time>            ::= a decimal number
<source-file>     ::= <file-ref>[;<file-ref>]...
<file-ref>        ::= <filedesc>1*
<memory-image>    ::= <filedesc>
<object-file>     ::= <filedesc>
<options>         ::= <option>[,<option>]...
<option>          ::= DEBU[G]
                    | ASCI[I]|BCD
                    | FORM|NFOR[M]
                    | LNO|NLNO
                    | NWAR[N]
                    | OPTZ|NOPT[Z]
                    | GO|NOGO
                    | ULIB|NOLI[B]|NULI[B]
                    | TIME=<number>
                    | CORE=<number>[K]
                    | TEST
                    | FDS|NFDS
                    | COMM[ON]=<number>
                    | NOBR[EAK]
                    | MAIN=<entry-name>
                    | MAP|NMAP
                    | SYMR[EF]
<user-library>    ::= <filedesc>[;<filedesc>]...
<run-time-file>   ::= <file-string>[;<file-string>]...
<file-string>     ::= <pseudo-file>|<cataloged-file>
<pseudo-file>     ::= <unit-number>|"<unit-number>"
<unit-number>     ::= two decimal digits
```

Description

<time> — is the maximum processor time in seconds that the program is allowed for execution.

<source-file> — is the set of file descriptors for FORTRANY source files in ASCII time-sharing-format, in the standard BCD card-image format, or in compressed card-image format (COMDK) and/or file descriptors for binary card-image object files. Alternatively, <source-file> may be a single file descriptor that contains a previously-generated system-loadable (H*) file.

A <source-file> consisting of the single character * indicates the current file. The <source-file> field is optional, and, if missing, indicates that only the current file is to be compiled.

<memory-image> is a single file descriptor of a random file into which the system-loadable (H*) file will be saved if the compilation and loading are successful, i.e., no fatal errors occur during compilation and loading. If the named file does not already exist, a permanent random file is created with an initial size of 36 LLINKS and general READ permission.

<object-file> A single file descriptor for a sequential file into which the compiler is to place the binary (C*) result of any indicated compilations. One object module is written to this file for each source program in the file(s)given as <source-file>. If the named object file does not already exist, a permanent sequential file is created with an initial size of 3 LLINKS and general READ permission.

<options> is a list of compilation and/or loading options.

All of the options are described below with the underlined options being the defaults.

DEBUG - The run time debug symbol table is generated for debugging.

<u>NDEBUG</u> - The run time symbol table is not generated.

<u>ASCII</u> - object character set is ASCII.

BCD - object character set is BCD. If this option is used, BCD must be specified whenever the General Loader is to be called. The General Loader is called for compile, compile-and-load, and load activities, but not for an execute-only run.

FORM - source is in "fixed" format; that is:

1) Source files may not have line numbers.

2) Comment lines are recognized by a C or * in character position 1.

3) Continuation lines are recoginzed by a nonblank, nonzero character in position 6.

4) Character positions 1 to 5 are reserved for statement labels.

5) Statements begin in character position 7 or thereafter.

Lines containing more than 72 characters have the additional characteristics:

6) Character positions 73-80 may be used for sequence identification information.

7) No more than 80 characters will be processed.

NFORM - source is in "free" format; that is:

1) Source files may or may not have line numbers.

2) Comment lines are recognized by a C or * in character position 1 for files without line numbers or as a C or * immediately following the line number for files with line numbers.

3) A continuation line is indicated by the ampersand character (&) as the first nonblank character of the line for files without line numbers or as an & as the first nonblank character following the line number for files with line numbers.

4) Character positions 73-80 may be used for sequence identification information for non-line-numbered files only. Statements may extend into these positions for line-numbered files.

LNO - source file has line numbers. This option only applies to NFORM files.

NLNO - source file does not have line numbers. This option only applies to NFORM files.

NWARN - turns off all the warning messages generated by a compilation and/or loading as long as no fatal errors occur.

OPTZ - optimize the object module.

NOPTZ - do not optimize the object module.

GO - the program will be executed at the completion of compilation.

NOGO - the program will not be executed at the completion of the compilation and/or loading. If an object file is specified, the object will be saved. If a memory image is specified, the system loadable (H*) file will be saved. If a memory image is not specified, only the compilation will be performed.

ULIB - file descriptors follow the <options> field and specify user libraries to be searched before the system library.

<u>NOLIB</u> - no user libraries are to be used.

<u>NULIB</u> - same as NOLIB.

TIME - <number>[K] - sets the time limit to the number of seconds the batch compilation and/or General Loader activity is to take, where <number> is less than or equal to 180. If not specified, the time is set to 60 seconds.

CORE=<number>[K] - sets the limits for the amount of memory to be used for compilation and loading. K represents 1024 words of memory. In the cases where K is specified or assumed, the number of words of memory requested is <number> multiplied by 1024.

For compilation:

Core specified	Core used
01 <= <number> <= 16	24K
17 <= <number> <= 40	<number>+8K
41 <= <number>	48K

For loading:

Memory specified	Memory used
01 <= <number> <= 40	<number>+8K
41 <= <number>	48K

Note: If (01 <= <number> <= 64) and the K is not specified, the number of words of memory requested is <number> times 1024.

If (<number> => 65) and the K is not specified, the number of words of memory requested is <number>.

If (<number> => 65) and the K is specified, the number of words of memory requested is <number> times 1024.

TEST - a test version of the compiler and/or General Loader is to be used for the batch activity. There must be an accessed fi.e.,(i.e., one in the AFT) of the name FORTRANY.

FDS - calls the FORTRAN DEBUGGING SYSTEM (FDS). FDS provides a symbolic dump, interactive debugging, timing- measurement, and user-supplied wrapup procedures.

NFDS - does not call the FORTRAN DEBUGGING SYSTEM (FDS).

COMMON=<number> - causes blank COMMON to be lowloaded. <number> must equal the total number of words of storage needed for blank COMMON. Normally, a user does not need this option.

NOBREAK - sets up a special program termination process to be invoked when the BREAK key is pressed during execution of the program.

This special termination process includes the normal emptying of buffers and closing of files. It does not include any calls to user-supplied wrapup routines or FDUMP (even if the FDS option was specified during compilation). This option is used when creating a core-image file for production use.

MAIN=<entry-name> - sets the program entry point for the program to be <entry-name>. If necessary, the module known by <entry-name> will be loaded from the libraries specified.

MAP - causes a memory map to be produced by the loader. A permanent file with an alternate name of P* should be accessed before the FRN command is issued. A reasonable size for this file is 24 LLINKS. After the FRN command is complete, the file "P*" will contain the load map in standard system format BCD.

NMAP - no memory map is produced.

SYMREF - turns on the SYMREF option for the loader. The "P*" file should be large if this option is used.

<user-library> a list of the names of files containing user libraries to be searched prior to the system library.

<run-time-file> - a list of file names which will be required during execution. The file names are separated by semi-colons. The file names may be in any of the following formats:

1. <pseudo-file> specifying a file name in the form of <unit-number> or "<unit -number>" where 01 <= <unit-number> <= 43. <unit-number> represents a logical unit referenced by the I/O statements in the program.

2. <cataloged-file>

   The user must specify an alternate name after the cataloged file name. The alternate name is the logical unit attached to the specified file, where 01 <= <alt-name> <= 43.

   If the <run-time-file> is a unit-number, a temporary file will be created for the user unless a file with the same name is directly under the user-ID.

   If the <run-time-file> is a unit-number specified in quotes, I/O will be directed to the terminal.

   Unit numbers 05, 06, 41, 42 and 43 are implicitly defined for terminal-directed I/O and need not be mentioned in the FTN command unless I/O is to be directed to a file. It is a good practice to use these unit numbers only for the files they represent by default.

Discussion

A user can include the FRN command as the first line or lines of his source file, subject to the following restrictions:

1. This feature is available on ASCII time sharing format files only.

2. The line or lines may be in the current file or a referenced permanent file; however, they must begin with the first line of the first source file.

3. The first two characters following the line number or for non-line-numbered files, the first two characters must be *#.

4. Multiple *# lines may appear in a source file, provided the total number of characters does not exceed 240.

5. The FRN command may be continued on more than one line. But, continuation to another line is only permissible when the preceding line ends with the delimiter ";" in a <source-file> list, a <user-library> list, and/or a <run-time-file> list as described in the General Form above.

6. The line(s) are treated as comment line(s) by the FORTRANY compiler.

7. The user can override a first-line FRN command by indicating save files or options on the FTN command to execute the file.

For example, a source program in the file S.PROG contains:

10*FRN S.PROG=(NWARN,ULIB)LIB/RTNS

At the time the file S.PROG is to be executed, suppose the user types the following command:

FRN S.PROG

The source file S.PROG is compiled, loaded, and executed. No warning messages are printed if compilation is successful. The user library LIB/RTNS is searched to resolve external references.

or, suppose the user types the following command:

FRN S.PROG=(NOGO)

The source file S.PROG is compiled and all warning messages are printed. Note, the first-line FRN command is overridden.

When a BCD or COMDK source file is supplied, the source file may also include an alter file descriptor in BCD format. The alter file must begin with a $ UPDATE card and must be in alter-number sequence. If more than one BCD or COMDK source file is specified, the alter file will update only the first.

Examples

1) FRN
(compiles, loads, and executes the current file.)

2) FRN
(same as FRN).

3) FRN P.SOURCE
(compiles, loads, and executes the FORTRANY source program in file P.SOURCE.)

4) FRN S.MAIN;S.SUB1;O.SUB2
(compiles source programs S.MAIN and S.SUB1, then binds them with the previously saved object file O.SUB2, loads, and executes.)

5) FRN S.PROG=HSTAR;CSTAR(ULIB)LIB/RTNS#IN"01";OUT"02"
(compiles, loads, and executes source program S.PROG. The core image will be saved on file HSTAR and the object on file CSTAR. For the execution, the random user library LIB/RTNS will be scanned to resolve external references such as subroutines and functions. The unit numbers 01 and 02 have been specified as alternate names for the files IN and OUT.)

6) FRN #"10"
(compiles, loads, and executes the current file and causes I/O for logical file code 10 to be directed to the terminal.)

7) FRN =H.PROG(CORE=30K,ULIB,NOGO)LIB/SPECIAL
(compiles and loads the current file and saves the core image on the random file H.PROG. 30K is specified for compilation and loading. The user library LIB/SPECIAL will be scanned to resolve external references. The program will not be executed.)

8) FRN H.PROG#02
(executes a previously saved core image that is in file H.PROG. The file 02 is accessed for I/O. If no such file exists in the user-ID, a temporary file with name 02 is created.)

9) FRN S.CAN=;O.CAN(NOGO)
(compiles the source program S.CAN and saves the object in file O.CAN.)

10) FRN LIBRARY/METRIC
(executes the previously saved core-image METRIC program that is stored in user-ID LIBRARY.)

Purpose

To access the file or files specified and place the file names in the Available File Table (AFT).

Format

GET <filedesc>[;<filedesc>]...

Discussion

If permissions are specified, they are used in accessing the file.

If permissions are not specified, and <filedesc> specifies a user-ID other than the current "LOGON" user-ID, the file access is attempted with general READ permission, and file allocation is subject to the permissions given to the file.

If permissions are not specified, and the file is in the current "LOGON" user-ID, the file is accessed with general READ and WRITE permission.

Under time sharing, a file name must be 8 characters or less. Thus, to access a file whose name is greater than 8 characters, a GET command can be issued specifying an alternate name of 8 characters or less. The alternate name is entered in the Available File Table. The file can then be referenced using that alternate name during the current "LOGON" session unless the alternate name is removed from the AFT with a REMOVE command. A linked file may be accessed in a random fashion by specifying MODE/RANDOM/ or M/R/ following the file description.

Examples

1) GET INPUT

2) GET INPUTFL,R

3) GET YOURID/GENRAL,R

4) GET MYID/EXPLANATION"ALTNAM",R

5) GET TEST/SEQ,MODE/RANDOM/

Purpose

The HELP command calls the HELP subsystem into use to assist in analyzing or explaining standard time sharing error messages.

Format

HELP

Discussion

The HELP subsystem gives the user further explanation of some of the error messages that are generated by various time-sharing subsystems and by the Time-Sharing System Executive.

These messages are prefixed by a number that can be used in requesting further explanation.

Refer to a later section for details of the HELP subsystem.

Examples

HELP

Purpose

The HOLD command prevents messages sent by the computer center or the MAIL command from appearing at the terminal.

Format

HOLD

Discussion

Messages issued by the computer center or the MAIL command are sent to the user's terminal after the user's next carriage return unless the AUTOMATIC command is in control or a HOLD command is in effect. If the user is in AUTOMATIC mode, the message is transmitted upon exit from AUTOMATIC mode.

If a HOLD is in effect, messages will be stopped until a SEND command is typed. The user must assume responsibility for any warning messages that are missed during the time of a HOLD. When a SEND is issued, the last message held is sent, any others are lost, and the HOLD is no longer in effect.

HOLD is used primarily before starting an interactive session to be used for display or reproduction purposes. The RUNOFF subsystem automatically puts HOLD on before starting and turns it off when finished.

Example

HOLD

Purpose

The IIDS command invokes the DM-IV Interactive I-D-S/II subsystem.

Format

IIDS

Discussion

DM-IV Interactive Integrated Data Store/II (DM-IV I-D-S/II) is a time sharing subsystem facility of the GCOS Data Management IV (DM-IV) System which allows a data base to be accessed through a remote terminal. Individual Data Manipulation Language (DML) statements similar to COBOL-74 can be entered, and any resultant currency and status register information can be obtained. Many data base features available in the batch version of DM-IV I-D-S/II are available from a remote terminal.

Purpose

The JABT command allows a user to abort a batch job submitted by the current user-ID.

Format

JABT[<snumb>]...

<snumb> ::= a batch job identification

Discussion

The current "LOGON" user-ID must be the same as the one specified on the $ USERID card of the job to be aborted.

A batch job cannot be aborted until it has reached a certain stage in GCOS job flow. Jobs that are in System Input (GEIN) or scheduler (RGIN) cannot be aborted until these phases are passed. A job may be aborted during any of the following stages: peripheral allocation, memory allocation, execution or termination. The system will respond with a "<snumb> not aborted - try again" message if the job has not reached the allocation stage.

Examples

1) JABT * (aborts most recent job submitted)

2) JABT 1234T

3) JABT 1234T 2345T

Purpose

The JDAC command establishes Direct-Access Communication (DAC) with a batch slave program.

Format

JDAC[<snumb>]

<snumb> ::= the snumb or program name of the DAC slave program

Discussion

The JDAC command is used when a user wants to establish conversational input/output between a terminal and a running batch job. When the command is given, the communication line is switched from time sharing to Direct Access. The terminal waits until the program specified by <snumb> connects to it. If there is no program called <snumb>, the terminal will continue to wait until the phone-line is disconnected. Since the communication line is no longer connected to time sharing, it is not possible to break out of Direct-Access mode while waiting for the program to connect to the terminal. The only way to get back to Time-sharing is to disconnect the terminal and re-establish the connection.

A batch job that can do conversational input/output with a terminal must contain a $ DAC card and the user-ID of the batch job must be validated for TALK permission by the central computer site.

If the batch job is submitted through the TSS batch interface (e.g. JRN), TALK disposition may be specified at the time of the run. This is the same as issuing a JDAC command.

Examples

1) JDAC 1234T

Purpose

The JOUT command invokes the JOUT subsystem to analyze the output reports produced by a user-submitted batch job. The output produced must have been generated using the same user-ID as that used during the LOGON interaction.

Format

JOUT [*|<snumb>]

<snumb> ::= a batch job identification
* ::= indicator that the last generated SNUMB (if present) is to be used to access output.

Discussion

The JOUT subsystem manipulates output from the following types of batch jobs:

- o those submitted through TSS batch interface via the JRN command with JOUT for a disposition.
- o those submitted through GRTS remote-facility Batch-entry or those submitted at the host that designate output is directed to a remote-station ID (e.g., $ REMOTE AA).

Upon entering JOUT if the job is still running, JOUT will print its status and return.

If the job has been released from the system (either at user request or because it has been printed), JOUT responds "OUTPUT NOT FOUND". It may also say this when the system output writer is not in memory, in which case the request should be tried again.

The message "OUTPUT BUSY" is displayed if another terminal has the same job in JOUT or if the job is printing at a remote or host.

JOUT types "FUNCTION ?" when it has made a connection to the batch output and is ready to accept commands. Possible functions are as follows:

ACTIVITY _n_. (ACTI)

JOUT prepares to read the activity specified by _n_ where _n_ cannot exceed 17.

DIRECT _id_ (DIRE)

Direct the output to the remote station specified by _id_.

DIRECT ONL (DIRE)

Print the output at the host.

EPRINT rc (E'RI)

Simulate printer report output. The report code (rc) may be any of the codes received from the LIST command, or $$ may be substituted for a report code. The $$ causes the printing of the J* file (control card list and execution report) at the terminal. Trailing blanks and blank lines are suppressed.

HOLD

If the user responds HOLD the subsystem deaccesses the SYSOUT data and returns the user to either the build mode or the subsystem level and the output is not processed by SYSOUT Report Writer. The output may subsequently be re-accessed by JOUT or may be manipulated by the host operator console verbs (e.g., PURGE).

KILR rc

Prevents printing of unwanted reports. The report code (rc) can be any of the codes received from the LIST command, with the exception of $$. The $$ report cannot be killed with KILR.

LIST

List the report codes associated with the current activity.

PRINT rc

Simulate printer report output. The report code (rc) may be any of the codes received from the LIST command, or $$ may be substituted for a report code. $$ causes the printing of the J* file (control card list and execution report) at the terminal. Multiple blanks are suppressed by the PRINT command.

RELEASE (RELE)

Remove the output from the system. No output is produced beyond this point.

SCAN rc

Scan the job output with the report code (rc). The system requests FORM? From this point, the question/answer sequence and the facilities available are the same as for the SCAN subsystem with the exceptions noted below:

1. The following SCAN verbs are not available: BATCH, REM, REM text, and BYE.

2. Output in memory dump format may be scanned. (Answer DUMP to the FORM? question. There is no initial subsystem reponse to this answer; the EDIT? question appears immediately.)

3. DONE returns the user to the FUNCTION level.

4. P - subsystem initially responds with the number of Severity Level 3 errors detected as determined by the PL/I compiler. A typical error message:

   ERROR 261, SEVERITY 3 ON LINE 122# 000103

5. STAT n
   STAT nn,nn,...nn

   Repositions the file to print the source statement in error.

Within JOUT multiple responses can be supplied on a single line, separated from one another by a space character. For example:

```
*JOUT 1234T SCAN 74 G YES ERRORS
```

NOTE: n consecutive spaces represent the equivalent of n-1 null responses.

Examples

```
1)  JOUT          (prompts for the snumb)
2)  JOUT 1234T
3)  JOUT *        (attempts access of output for the last generated SNUMB)
4)  JOUT * SCAN 74 G Y E DONE
```

Purpose

The JPRINT and JPUNCH commands are the CONVERT commands used to initiate host jobs to produce printer and punch output for the designated input file.

Format

JPRI[NT][infile][:options]

```
[infile]     ::= [filedescr]
[:options]   see below.
```

Description

Media Code Options

The output record format options specify the physical format of the output record. The default option for the CONVERT command is "ASCII". A list of the options and their meanings is as follows:

BCD - variable length BCD - media code 0

COMDK - BCD compressed deck card image (COMDK) - media code 1

CARD - BCD 14-word card image - media code 2

PRINT - BCD variable-length print line image - media code 3

OLDASC - obsolete TSS ASCII - media code 5

ASCII - standard system format ASCII - media code 6

APRINT - ASCII print line image - media code 7

ACARD - ASCII card image - media code 10

SAME - a record output media code is the same as its input media code

Line Number Options

Line numbers can exist with COMDK, CARD, ACARD, OLDASC, and ASCII records. All BCD, PRINT, and APRINT records cannot possess line numbers. The line number for an ASCII or OLDASC record consists of 1 to 8 numeric characters. These numeric characters must be among the first eight characters in a line. A line number is defined to include any leading blanks. A line number is terminated by a nonnumeric character, including blank. If the "#" character terminates a line number and if it is one of the first eight characters of a line, it is considered to be a delimiter. It is treated as neither part of the line number nor part of the text. The line number for COMDK, CARD, and ACARD records is defined to be all the trailing digits in columns 73-80. This field may begin with nonnumerics; these also are considered neither part of the line number nor part of the text.

The line number options may specify:

1. Whether line numbers are to appear in the output text.

2. The actual line number values.

The default line number option is "ASIS". A description of each of the options follows:

ASIS — Line numbers are assumed not be present in the input file. Text, including leading/trailing numeric characters and "#"'s are left as is.

STRIP — Strip line numbers from the input text before reformatting and writing the output text. Input COMDK, CARD, and ACARD records are truncated at column 72. Line numbers on ASCII and OLDASC records, when present, are discarded and the first character following the line number is treated as the first character of the line.

MOVE — Move line numbers. The input records have the line numbers detached from the text string, either from the front (ASCII or OLDASC) from columns 73-80 (COMDK, CARD, or ACARD). The output records have the line numbers reattached to the text string, either at the front (ASCII or OLDASC) or in columns 73-80 (COMDK, CARD, or ACARD). If the output records are BCD, PRINT, or APRINT, the line numbers are not re-attached and the M option acts similar to the S option.

I(i,j) — Insert line numbers beginning with line number i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. The input file is assumed not to be line-numbered. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the I option is ignored.

R(i,j) — Resequence line numbers. Strip any existing line numbers from the input text and insert new line numbers in the output text, beginning with i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the R option behaves as the S option.

N(ch) — Implies the M option and specifies that the normal tab character (the colon) and tab settings (8, 16, 32, 73) have been employed in building the input file(s). The (ch) argument may be used to define a character which replaces the colon as the tab character.

LABEL (abcde(i-j)fghi(i-j)---) If the output records are COMDK, CARD, or ACARD, then a label is placed left-justified in columns 73-77. The label is specified as 1 to 5 nonblank characters. The fields "abcde" and "fghij" represent the labels. The label is placed on only those lines with line numbers between i and j inclusive. Up to 10 distinct labels may be given. If more than one label is given though, the (i-j) specifications may not overlap.

The LABEL option is meaningful only if line numbers are attached to output records. Therefore, the label option is completely ignored unless it is accompanied by either the insert, resequence, or move option.

For the I and R options, output line numbers for ASCII and OLDASC records will have at least the number of digits specified for i in I(i,j) or R(i,j). Thus R(0010,10) will result in line numbers 0010, 0020, 0030,---.

Input records are assumed to have line numbers when the STRIP, MOVE, and RESEQUENCE options are specified. Otherwise, line numbers are assumed to be absent and leading numerics in ASCII format are treated as real text. When line numbers are assumed present, tabbing and columnizing are performed relative to the start of the real text.

The user must be careful not to alter the line number values of a BASIC file.

Character Manipulation Options

A description of each of the character manipulation options follows.

TAB(ch,i,j---;ch,i,j---;----) Expand tab characters into blanks. Use "ch" as a tab character with settings i,j,k,etc. Usually, any occurrence of the tab character in the input file(s) results in the replacement of the tab character with a string of blanks up to the next tab setting. However, if a tab character is encountered beyond the last tab setting specified for that tab character, it is treated as a normal non-tab character.

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the tab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

UNTAB(ch,i,j---;ch,i,j---;----) Insert tab characters, replacing blanks. Use "ch" as a tab character with settings i, j, k, etc. Any occurrence of a string of blanks terminating on an "untab" tab stop is replaced by the character "ch".

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the untab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

LOWER Convert all alphabetic characters to lowercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

UPPER — Convert all alphabetic characters to uppercase. This option is meaningful only if the output records are ASCII, OLDSAC, or APRINT.

BEGIN(ch) — Begin a new line (record) immediately after the character "ch". The character "ch" is treated as a delimiter and not part of the text. It is not placed in the output text. When the "ch" character is located at the beginning or end of a line, it is simply deleted. Strings of the "ch" characters are treated as a single "ch" character.

COLUMNS(i-j) — Delete all of the characters in a line except those which are located within columns i through j inclusively. The options BEGIN and TAB are both completed before COLUMNS takes effect. If a record does not extend through column j prior to the COLUMNS option execution, it is blank filled to column j. Thus, when the COLUMNS options is in effect, the length of all generated output records is j-i+1 characters.

SQUEEZE — Replace any string of two or more blanks by a single blank. The options BEGIN, TAB, COLUMNS, and UNTAB are all performed before SQUEEZE is executed.

TRAIL — Delete all trailing blanks on a line. The TRAIL option is performed immediately after the SQUEEZE option.

A number of options affect the length of an output text line. It is important that the user understand the order in which these options are performed. The order (from first to last) in which the options are executed is:

BEGIN
TAB
COLUMNS
UNTAB
SQUEEZE
TRAIL

Miscellaneous options

VERIFY — If the VERIFY option is in effect when CONVERTY completes the processing of an input file, then CONVERT gives a brief summary of the number of records obtained from the file. This summary gives, for each media code, the number of records which had that media code.

IGNORE — Ignore all embedded $$ control lines. Treat them as text.

DISCARD — Discard all nontext records. Nontext records are those records whose media code is not one recognized and interpreted by CONVERT. The JRN, JPRINT, JPUNCH, APRINT, and DISPLAY commands require that nontext records be discarded. The CONVERT command normally does not require that nontext records be discarded. When nontext records are encountered during the execution of the CONVERT command, they are written to the output file, but no reformatting or media conversion is performed.

JPRINT
JPUNCH

TIME When the TIME option is invoked, the date and time of day are printed at the user's terminal.

DEFAULT The D'FAULT option is used to nullify all options which the user has sp'cified either on the command line or embedded $$ control lines. The default option has no affect on any of the "specialized" options. Because of the nature of the DEFAULT option, it is meaningless for it to be located in the options field of the command line. Therefore, if the DEFAULT option is encountered in the options field, an error message is issued. The same reasoning applies to the placement of the DEFAULT option anywhere other than the beginning of a $$ control line.

File Processing Options

SELECT (file) The SELECT option is analogous to the $ SELECTA card. The select option allows an input file to specify other input files. Upon encountering the SELECT option, the selected file is obtained and is used in place of the $$ control line. Nesting of selects is permitted up to 17 levels. The SELECT option is meaningful and valid only on a $$ control line. Only one SELECT option may be specified on a $$ control line.

INCLUDE If the INCLUDE option is in effect, CONVERT, upon encountering the SELECT option, uses the selected file as an input file.

EXCLUDE If the EXCLUDE option is in effect, CONVERT ignores the SELECT option.

The purpose of the INCLUDE and EXCLUDE options is to allow the user to control the performance of the select options while not forcing him to disregard:

1. Other options on the same $$ control line.

2. All $$ control lines.

The INCLUDE option is the default option for the JRN command. The EXCLUDE option is the default option for the JPRINT, JPUNCH, APRINT, DISPLAY, and CONVERT commands.

Specialized Options

The "specialized" options are a class of options completely distinct and separated from all preceding options. The "specialized" options are unlike other options in that they take effect only when all input files have been read, converted, and closed; i.e., after the output file has been completely generated. All other options, of course, are meant to be used when the output file is in the process of being generated.

ROUT(xx)	The ROUT option is applicable to the JRN, JPRINT, APRINT and JPUNCH commands. This option causes the implied files generated by the program execution to be directed to the specified two-character remote station. Only one ROUT entry is permitted.
WAIT	The WAIT option is applicable to the JRN, JPRINT, APRINT and JPUNCH commands. This option causes the user to wait until the completion of the spawned job in the batch environment. The wait period may be broken out of by hitting the break key. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem in invoked.
COPY(nn)	The COPY option is applicable only to the the JPRINT, APRINT and JPUNCH commands. this option causes the generation of nn multiple copies of the listing or punched deck. The maximum number of copies that can be produced from a single JPRINT/JPUNCH job is 13.
IDENT(info)	The IDENT option is applicable to the JPRINT, APRINT, and JPUNCH commands. This option allows the user to minimize the subsystem/user interface involved in the use of the JPRINT/JPUNCH commands. When the IDENT option is present, the normal question/answer sequence of $ IDENT? response is bypassed. The information presented as the IDENT option argument is used instead of the user-response to the question.
MONITOR	The MONITOR option is applicable to the JPRINT, APRRINT, JPUNCH, and JRN commands. This option allows the user to monitor or track the status of his spawned job as it is executed in the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.
DIRECT	The DIRECT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. If the DIRECT option is given on the command line, it overrides any JOUT or ROUT option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the JOUT option to place it on a $$ control line. He can then override it without being required to change his $$ control line.

The ROUT, JOUT, and DIRECT options are mutually exclusive. The MONITOR, TALK, WAIT, and DISMISS options are also mutually exclusive. Mutually exclusive options are a group of options for which only one member of the group of options may be in effect. If the user attempts to give two mutually exclusive options in the options field of the command line or an a $$ control line, an error message is given.

Discussion

The printing or punching is done at the host.

Examples

```
JPRINT FILEA(:T,S)
```

The contents of FILEA are printed at a high-speed printer. Tabs are expanded and line numbers are stripped. The tab character is : and the settings are 8, 16, 32, and 73.

Purpose

The JRN command causes a job under control of the CONVERT subsystem to "RUN" as a batch processing job.

Format

```
JRN [infile(s)][=otfile][:options]

   [infile(s)] ::= *|**|[filedescr]
   [=otfile]   ::= *|=**|[=filedescr]
   [:options]  see below
```

Description

Media Code Options

The output record format options specify the physical format of the output record. The default option for the CONVERT command is "ASCII". A list of the options and their meanings is as follows:

BCD - variable length BCD - media code 0

COMDK - BCD compressed deck card image (COMDK) - media code 1

CARD - BCD 14-word card image - media code 2

PRINT - BCD variable-length print line image - media code 3

OLDASC - obsolete TSS ASCII - media code 5

ASCII - standard system format ASCII - media code 6

APRINT - ASCII print line image - media code 7

ACARD - ASCII card image - media code 10

SAME - a record output media code is the same as its input media code

Line Number Options

Line numbers can exist with COMDK, CARD, ACARD, OLDASC, and ASCII records. All BCD, PRINT, and APRIN records cannot possess line numbers. The line number for an ASCII or OLDASC record consists of 1 to 8 numeric characters. These numeric characters must be among the first eight characters in a line. A line number is defined to include any leading blanks. A line number is terminated by a nonnumeric character, including blank. If the "#" character terminates a line number and if it is one of the first eight characters of a line, it is considered to be a delimiter. It is treated as neither part of the line number nor part of the text. The line number for COMDK, CARD, and ACARD records is defined to be all the trailing digits in columns 73-80. This field may begin with nonnumerics; these also are considered neither part of the line number nor part of the text.

The line number options may specify:

1. Whether line numbers are to appear in the output text.

2. The actual line number values.

The default line number option is "ASIS". A description of each of the options follows:

ASIS Line numbers are assumed not be present in the input file. Text, including leading/trailing numeric characters and "#"'s are left as is.

STRIP Strip line numbers from the input text before reformatting and writing the output text. Input COMDK, CARD, and ACARD records are truncated at column 72. Line numbers on ASCII and OLDASC records, when present, are discarded and the first character following the line number is treated as the first character of the line.

MOVE Move line numbers. The input records have the line numbers detached from the text string, either from the front (ASCII or OLDASC) from columns 73-80 (COMDK, CARD, or ACARD). The output records have the line numbers re-attached to the text string, either at the front (ASCII or OLDASC) or in columns 73-80 (COMDK, CARD, or ACARD). If the output records are BCD, PRINT, or APRINT, the line numbers are not re-attached and the M option acts similar to the S option.

I(i,j) Insert line numbers beginning with line number i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. The input file is assumed not to be line-numbered. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the I option is ignored.

R(i,j) Resequence line numbers. Strip any existing line numbers from the input text and insert new line numbers in the output text, beginning with i and incrementing by j. The arguments i and j are optional. If they are not given, the defaults are i=10 and j=10. If the output records are BCD, PRINT, or APRINT, line numbers are not inserted and the R option behaves as the S option.

N(ch) Implies the M option and specifies that the normal tab character (the colon) and tab settings (8, 16, 32, 73) have been employed in building the input file(s). The (ch) argument may be used to define a character which replaces the colon as the tab character.

LABEL (abcde(i-j)fghi(i-j)---) If the output records are COMDK, CARD, or ACARD, then a label is placed left-justified in columns 73-77. The label is specified as 1 to 5 nonblank characters. The fields "abcde" and "fghij" represent the labels. The label is placed on only those lines with line numbers between i and j inclusive. Up to 10 distinct labels may be given. If more than one label is given though, the (i-j) specifications may not overlap.

The LABEL option is meaningful only if line numbers are attached to output records. Therefore, the label option is completely ignored unless it is accompanied by either the insert, resequence, or move option.

For the I and R options, output line numbers for ASCII and OLDASC records will have at least the number of digits specified for i in I(i,j) or R(i,j). Thus R(0010,10) will result in line numbers 0010, 0020, 0030,---.

Input records are assumed to have line numbers when the STRIP, MOVE, and RESEQUENCE options are specified. Otherwise, line numbers are assumed to be absent and leading numerics in ASCII format are treated as real text. When line numbers are assumed present, tabbing and columnizing are performed relative to the start of the real text.

The user must be careful not to alter the line number values of a BASIC file.

Character Manipulation Options

A description of each of the character manipulation options follows.

TAB(ch,i,j---;ch,i,j---;----) Expand tab characters into blanks. Use "ch" as a tab character with settings i,j,k,etc. Usually, any occurrence of the tab character in the input file(s) results in the replacement of the tab character with a string of blanks up to the next tab setting. However, if a tab character is encountered beyond the last tab setting specified for that tab character, it is treated as a normal non-tab character.

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the tab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

UNTAB(ch,i,j---;ch,i,j---;----) Insert tab characters, replacing blanks. Use "ch" as a tab character with settings i, j, k, etc. Any occurrence of a string of blanks terminating on an "untab" tab stop is replaced by the character "ch".

If a tab character is specified without specifying any tab settings, default settings of 8, 16, 32, and 73 are assumed. If the untab option is given without any arguments, the normal tab character, colon, and the default settings are assumed. There is no limit to the number of tab characters or settings allowed.

LOWER — Convert all alphabetic characters to lowercase. This option is meaningful only if the output records are ASCII, OLDASC, or APRINT.

UPPER — Convert all alphabetic characters to uppercase. This option is meaningful only if the output records are ASCII, OLDSAC, or APRINT.

BEGIN(ch) — Begin a new line (record) immediately after the character "ch". The character "ch" is treated as a delimiter and not part of the text. It is not placed in the output text. When the "ch" character is located at the beginning or end of a line, it is simply deleted. Strings of the "ch" characters are treated as a single "ch" character.

COLUMNS(i-j) — Delete all of the characters in a line except those which are located within columns i through j inclusively. The options BEGIN and TAB are both completed before COLUMNS takes effect. If a record does not extend through column j prior to the COLUMNS option execution, it is blank filled to column j. Thus, when the COLUMNS options is in effect, the length of all generated output records is j-i+1 characters.

SQUEEZE — Replace any string of two or more blanks by a single blank. The options BEGIN, TAB, COLUMNS, and UNTAB are all performed before SQUEEZE is executed.

TRAIL — Delete all trailing blanks on a line. The TRAIL option is performed immediately after the SQUEEZE option.

A number of options affect the length of an output text line. It is important that the user understand the order in which these options are performed. The order (from first to last) in which the options are executed is:

BEGIN
TAB
COLUMNS
UNTAB
SQUEEZE
TRAIL

Miscellaneous options

VERIFY — If the VERIFY option is in effect when CONVERT completes the processing of an input file, then CONVERT gives a brief summary of the number of records obtained from the file. This summary gives, for each media code, the number of records which had that media code.

IGNORE — Ignore all embedded $$ control lines. Treat them as text.

DISCARD — Discard all nontext records. Nontext records are those records whose media code is not one recognized and interpreted by CONVERT. The JRN, JPRINT, JPUNCH, APRINT, and DISPLAY commands require that nontext records be discarded. The CONVERT command normally does not require that nontext records be discarded. When nontext records are encountered during the execution of the CONVERT command, they are written to the output file, but no reformatting or media conversion is performed.

TIME — When the TIME option is invoked, the date and time of day are printed at the user's terminal.

DEFAULT — The DEFAULT option is used to nullify all options which the user has specified either on the command line or embedded $$ control lines. The default option has no affect on any of the "specialized" options. Because of the nature of the DEFAULT option, it is meaningless for it to be located in the options field of the command line. Therefore, if the DEFAULT option is encountered in the options field, an error message is issued. The same reasoning applies to the placement of the DEFAULT option anywhere other than the beginning of a $$ control line.

File Processing Options

SELECT (file) — The SELECT option is analogous to the $ SELECTA card. The select option allows an input file to specify other input files. Upon encountering the SELECT option, the selected file is obtained and is used in place of the $$ control line. Nesting of selects is permitted up to 17 levels. The SELECT option is meaningful and valid only on a $$ control line. Only one SELECT option may be specified on a $$ control line.

INCLUDE — If the INCLUDE option is in effect, CONVERT, upon encountering the SELECT option, uses the selected file as an input file.

EXCLUDE — If the EXCLUDE option is in effect, CONVERT ignores the SELECT option.

The purpose of the INCLUDE and EXCLUDE options is to allow the user to control the performance of the select options while not forcing him to disregard:

1. Other options on the same $$ control line.

2. All $$ control lines.

The INCLUDE option is the default option for the JRN command. The EXCLUDE option is the default option for the JPRINT, JPUNCH, APRINT, DISPLAY, and CONVERT commands.

Specialized Options

The "specialized" options are a class of options completely distinct and separated from all preceding options. The "specialized" options are unlike other options in that they take effect only when all input files have been read, converted, and closed; i.e., after the output file has been completely generated. All other options, of course, are meant to be used when the output file is in the process of being generated.

MONITOR — The MONITOR option is applicable to the JPRINT, APRINT, JPUNCH, and JRN commands. This option allows the user to monitor or track the status of a spawned job as it is executed in the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

JOUT — The JOUT option is applicable only to the JRN command. This option results in all implied files being saved so that they may be examined using the JOUT subsystem.

ROUT(xx) — The ROUT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. This option causes the implied files generated by the program execution to be directed to the specified two-character remote station. Only one ROUT entry is permitted.

WAIT — The WAIT option is applicable to the JRN, JPRINT, APRINT and JPUNCH commands. This option causes the user to wait until the completion of the spawned job in the batch environment. The wait period may be broken out of by hitting the break key. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem is invoked.

TALK	The TALK option is applicable only to the JRN command. This option implies that the batch job includes execution of a program containing conversational (direct access) input/output. This option causes the user's terminal to be placed in direct access connection with the submitted program (by SNUBM) following its submission to the batch environment. When the job completes execution, the user is informed of the job's termination status and, if the JOUT option is in effect, the JOUT subsystem in invoked.
URGC(xx)	The URGENCY option is applicable only to the JRN command. This option indicates that the user wishes to assign initial urgency xx to the spawned batch job. If the assigned urgency is greater than the maximum allowed for the user, the message ILLEGAL URGENCY is sent and the batch job is not spawned. If xx is not specified, maximum allowable urgency is automatically assigned.
DIRECT	The DIRECT option is applicable to the JRN, JPRINT, APRINT, and JPUNCH commands. If the DIRECT option is given on the command line, it overrides any JOUT or ROUT option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the JOUT option to place it on a $$ control line and later override it without changing the $$ control line.
DISMISS	The DISMISS option is applicable only to the JRN command. If the DISMISS option is given on the command line, it overrides any TALK, WAIT, or MONITOR option which the user has placed on a $$ control line. This option allows the user who, for instance, usually specifies the MONITOR option to place it on a $$ control line. He can then override it without being required to change his $$ control line.

The ROUT, JOUT, and DIRECT options are mutually exclusive. The MONITOR, TALK, WAIT, and DISMISS options are also mutually exclusive. Mutually exclusive options are a group of options for which only one member of the group of options may be in effect. If the user attempts to give two mutually exclusive options in the options field of the command line or on a $$ control line, an error message is given.

Discussion

The three fields do not have to be ordered as shown; however, if the input file name is not the file name following the command it must be preceded by a semicolon.

Examples

```
JRN      INPUT

INPUT =  10$$S,¯,J,V,MONI
         20$:IDENT:M24GPCX13, JANEDOE , STATION G
         30$:OPTION:NOSETU,NOGO
         40$:LOWLOAD
         50$:GMAP:COMDK
         70$$ SELECT(PROJECT/SCHD(:T(;),V))
         80$$ SELECT (PROJECT/ALTER(:V,S))
         90$:LIMITS:,,,25000
         100$:PRMFL:K*,R/W,S,PROJECT/SCHD-COM
         110$:PRMFL:C*,R/W,S,PROJECT/SCHD-OBJ
         120$:ENDJOB
```

The program contained in INPUT is passed to the batch system along with the contents of the files PROJECT/SCHD and PROJECT/ALTER. INPUT has its line numbers stripped and its tabs expanded where the tab characters is : and the settings are 8, 16, 32, and 73. PROJECT/SCHD has its line numbers left ASIS and its tabs expanded where the tab character is ; and the settings are 8, 16, 32, and 73. PROJECT/ALTER has its line numbers stripped and no tabs are expanded. A report is given for all three files which gives the number and type of records obtained from each file. The program is given JOUT disposition and the execution of the program is monitored.

Purpose

To print the current processing status of the batch job or jobs specified by snumb number.

Format

JSTS[*|<snumb>]...

<snumb> ::= a batch job identification

* ::= indicator that the identifier of last job submitted is to be used

Discussion

The following are the status messages returned by JSTS and their meanings:

MESSAGE	MEANING
STATUS CHANGING	The job is in a transitional state.
READING-RMT	The job is being read by the batch system.
WAIT-ALLOC	The job is not yet a candidate for peripheral allocation .
WAIT-PERIP	The job is waiting for peripheral allocation.
WAIT-CORE	The job is waiting for core allocation.
IN HOLD	A hold was initiated by the operator (perhaps the job needed a tape or disk pack that was already being used) or the job includes a $PRIVITY card.
IN LIMBO	The job is waiting for the host operator to fetch tapes, disk packs, special forms, etc.
EXECUTING	The job is in execution.
SWAPPED	A job with higher urgency has caused your job to be temporarily swapped out of memory.
WAIT-MEDIA	The job is waiting completion of a tape or disk-pack mount request.
IN SIEVE	The job's resource requirements exceed the limits set by the host. The job will be run when the machine is less busy.
OVERDUE	The job has reached a high urgency but the system still has not been able to get resources necessary to run your job, i.e., disk packs, tapes, or permfiles are busy.

IN RESTART	The system is restarting your job after a service interruption.
TERMINATION	The activity has finished executing and is in the terminating procedure.
OUTPUT WAITING	Execution is complete but printing is not. At this time, JOUT output may be accessed.
OUTPUT COMPLETE	Printing, punching and remote I/O is complete.
JOB NOT ACCESSIBLE	The job is not yet far enough into the system to identify its status; the job is in sysout and sysout is swapped; or, the output is complete and the job is no longer in the system.

Early in the processing of a job, the Scheduler looks at the system resources the job requires and puts the job in an appropriate queue. If JSTS is requested when a job is in the Scheduler, a message of the following form is printed:

IN SCHEDULER <name> QUEUE

Where:

<name> is one of the queues defined by the site in the system startup definitions. For additional information see DRL JSTS description.

Examples

1) JSTS * (prints status of the most recent job submitted through TSS Terminal Batch-entry during the current "LOGON" session. If the job has terminated, the termination code is printed.)

2) JSTS 1234T

3) JSTS 4567W 5678T

LCAS
LEADER

Purpose

The LCAS command allows both uppercase and lowercase characters to be transmitted to a keyboard/display device.

Format

LCAS[E]

Discussion

This command applies to keyboard/display type devices only. See the UCAS command.

Purpose

The LEADER command causes a title to be punched in bold, block letters in the paper tape followed by a list of the current file.

Format

LEAD[ER] [title]

Discussion

If a title is not entered, the system requests the title. Although only uppercase characters are punched, the title can be composed of upper or lowercase alphabetics, numerics, and special characters except the commercial at (@) sign. The date is punched in the international standard format following the title. A series of ASCII characters (carriage return, line feed, and 8 rubouts) follows the date and preceeds the contents of the current file. After the current file has been punched another similar stream of ASCII characters (plus an X-OFF (DC3) character) is appended to the paper tape.

Example

```
* LEAD MY NAME
```

Purpose

The LENGTH command generates a report of the type, current length and content length of the specif ed file or files.

Format

LENG[TH][<file-ref>[;<file-ref>]...]

<file-ref> ::= *|<file desc>

Discussion

There are 3 lengths associated with every file:

1) the content (or used) length,

2) the current length,

3) the maximum length.

The lengths are measured in LLINKS, which are 320 word blocks.

The content length is the number of LLINKS used to store the contents of a file.

The current length is the number of LLINKS reserved on mass storage for a file. This is the amount the user is charged for. If the content length is smaller than the current length, the difference is being wasted.

The maximum length is the length a file can grow to. All software will try to grow a file if it needs more room, so it is best to make the maximum length greater than the current length to allow for growth.

Examples

1) LENGTH
   (prints the type and content length of the current file)

2) LENGTH *
   (prints the type, current length and content length of the current file)

3) LENG PROG1
   (prints the type, current length and content length of file PRPG1)

4) LENG *;PROG1
   (prints a length report for the current file and file PROG1)

5) LENG COLORS/PRIMARY
   (prints a length report on the file PRIMARY in the subcatalog COLORS)

6) LENGTH COUNTY/SERVICES,R
   (prints a length report on the file SERVICES in the user-ID COUNTY)

Purpose

The LIB command copies a program or a portion of a program from the Common Library to the curret file.

Format

```
LIB[ <lib-prog>[(<line-range>)]]

     <lib=prog>  ::= a program in the Common Library
     <line-range>::= <begin-line>-<end-line>
                   |  <begin-line>-
                   |  -<end-line>
     <begin-line>::=<line>
      <end-line> ::= <line>
      <line>     ::= a 1- to 8-digit decimal number
```

Discussion

There are many BASIC and FORTRANY programs in the Common Library including engineering, business, mathematical and statistical applications. The applications library is distributed in file system restorable format for restoration to user-ID LIBRARY.

Examples

1) LIB
   (prompts for the Common Libary routine name)

2) LIB LIMITS

3) LIB WINGDATA (100-9999)
   (picks up lines 100 through 9999 of program WINGDATA)

LINE
LINELENGTH

Purpose

The LINELENGTH command increases the maximum length of an input line that may be sent from a terminal.

Format

LINE[LENGTH]<nn>

<nn> ::= a number between 80 and 160.

Discussion

1) The LINELENGTH command cannot be used on VIP type terminals.

2) LINE or LINELENGTH followed by a carriage return will set the line length to 80 characters.

Examples

1) LINE (sets the line length to 80 characters)

2) LINE 120 (sets the line length to 120 characters)

Purpose

The LIST command lists the contents of the specified ASCII time sharing file, files or file segments.

Format

```
     LIST[H|E[<columns>]][  <file-list>|<line-list>]

or   LISTL[  *|<filedesc>]

          <columns>    ::= a decimal number
          <file-list> ::= <file-ref>[;<file-ref>]...
          <file-ref>   ::= [*|<filedesc>][(<line-list>)]
          <line-list> ::= <line-ref>[,<line-ref>]...
          <line-ref>   ::= <line>|<line-range>
          <line-range>::= <begin-line>-<end-line>
                        |= <begin-line>-
                        |= -<end-line>
          <begin-line>::= <line>
          <end-line>   ::= <line>
          <line>       ::= a 1- to 8-digit decimal number
```

Description

<line-ref> defines which lines and/or line ranges are to be listed. A combination of single line numbers and line-number ranges may be used. However, the line numbers selected must be in ascending order.

<begin-line> indicates which line number is to start the listing. The form <begin-line>- means "list to the end of the file" and can only be used as either the first-and-only or the last line range.

<end-line> indicates the line number at which a line range ends. The form -<end-line> means "list from the beginning of the file" or, if -<end-line> is preceded by any lines or line ranges, list starting at the line after the last one already listed."

Discussion

If no file name is specified in either form of the LIST command, the current file is assumed. In both forms, the current file may be specified explicitly by an asterisk.

If no line numbers are specified in the first form of the LIST command, the entire contents of the file are listed. With the LIST command, the contents of a file or files with general READ permission in another user-ID can be listed. The current file is never altered by the LIST command even when other files or file segments are listed.

LISTH precedes the listing with a header that contains the date and time.

LISTE<columns> lists the file but with all lines "broken" or "folded" at the character position specified by <column>. In Example 12 if PARTS were a file that contained an 80-character line, each line in the file would be listed as 4 lines. If no <columns> value is given, the default value of 72 is used. Some terminals try to fold lines that exceed the carriage width, but some characters are usually dropped. To ensure that lines that exceed the carriage width will be listed and folded properly, the LISTE command should be used.

LISTL lists the last line in the file. If no file name or an asterisk is specified, the last line in the current file is listed. If a file name is provided, the last line in that file is listed.

Examples

1) LIST
(lists the entire contents of the current file)

2) LIST *
(same as LIST)

3) LIST 50-80,90,120
(lists lines 50 through 80, 90, and 120 of the current file)

4) LIST 500-
(lists from line 500 through the end of the current file)

5) LIST 60,100,300-
(lists lines 60, 100, and 300 through the end of the current file)

6) LIST -50
(lists from the beginning of the current file through line 50)

7) LIST 150-180,400-420
(lists lines 150 through 180 and lines 400 through 420 of the current file)

8) LIST /AIRCRAFT/TANKERS
(lists the entire contents of the file TANKERS in subcatalog AIRCRAFT)

9) LIST PLASTIC(10-50,100-150);METAL(10,30,200-)
(lists lines 10 through 50 and lines 100 through 150 of file PLASTIC; and lists lines 10, 30, and 200 through the end of the file METAL)

10) LISTH WOODTYPE
(lists the entire contents of file WOODTYPE with a date and time header)

11) LISTH
(lists the entire contents of the current file with a date and time header)

12) LISTE25 NAILSIZE
(lists the entire contents of the file NAILSIZE with each line in the file listed with 25 columns per line on the terminal)

13) LISTE /CUSTOM/FURNITURE
(lists the entire contents of the file FURNITURE in subcatalog CUSTOM with each line in the file listed with 72 columns per line on the terminal)

14) LISTL
(lists the last line of the current file)

15) LISTL *
(same as LISTL)

16) LISTL /MAKER/TOOLS
(lists the last line of the file TOOLS in subcatalog MAKER)

17) LIST USERID/ROSTER
(lists the entire contents of the file ROSTER in user-ID USERID)

Purpose

The LODS command loads a specified TSS subsystem which is bound with the trace package.

Format

```
LODS <subsys>[:inputdata]

    <subsys>    ::= name of a TSS subsystem
    <inputdata> ::= <text string parameters to pass to subsystem>
```

Discussion

LODS is similar to LODT, except that a specified TSS subsystem, instead of an H* file, is loaded and bound with the trace package. This capability is primarily intended for those responsible for subsystem maintenance and site system personnel. The command associated with the desired subsystem, followed by any of its necessary parameters may accompany the LODS command. If not specified on the same line as the LODS command, this information is requested from the user. As with LODX and LODT, an opportunity is given to "PATCH, SAVE (filedescr required) OR RUN". Prior to relinquishing control, LODS removes all characters of the input line that prefix the command word (via DRL PSEUDO). This would normally be the LODS command itself and its terminating delimiter. Thus, for example, the following use of LODS would result in loading the LIST subsystem for debugging purposes:

```
LODS LIST FILEX(100,220);FILEY
```

Use of LODS for debugging privileged subsystems must be requested at system selection level and is permitted for the master user only. Such subsystems include LOGOFF, TERM, NEW, NEWU, JSTS, JOUT, and LODS itself. Master subsystems cannot be debugged with LODS.

The LODT and LODS commands permit the load origin of the Trace Package to be specified. (See the <u>Debug and Trace</u> manual.) This specification must be preceded by a semicolon and requires the format, TRACE-nnnnnn, where nnnnnn is the desired octal address at which to load the Trace Package. Thus, to load a program with the Trace Package origin at location 14000:

```
*LODSJOE/JSTS;TRACE-14000:JSTS1234T
```

Examples

```
1)  LODS  HELP

2)  LODS  ACCE:ACCE LS,/TEMP
```

Purpose

The LODT command loads a user-supplied program from the H* file and appends a copy of the trace package.

Format

LODT <infile>

<infile> ::= <filedescr>

Discussion

The LODT subsystem provides a debugging environment for a user program resident on an H* file. As with LODX, the H* file is loaded and the user given the opportunity to "PATCH, SAVE OR RUN". In addition, however, a copy of the trace package is appended to the resulting load, and TRACE is provided with the program's true entry address in its linkage register (X1). When the RUN command is given, LODT transfers control to the trace package. TRACE is thus initially given control, and when its first "R" command is exercised, program execution begins. If the trace mechanism is engaged before issuing the "R" command, the user's program will be executed in a controlled environment.

See also the later section on debugging subsystems for PATCH, SAVE and RUN functions.

Examples

1) LODT MYID/HSTAR;TRACE-2000

2) LODT QUIKFILE

3) LODT SUBSYS:INPUT

Purpose

The LODX command loads a user supplied program from an H* file.

Format

LODX <infile>

<infile> ::= <filedescr>

Discussion

Upon completion of the loading function the user receives the message

PATCH, SAVE OR RUN?

Only the first character (P, S, or R) is necessary. If a null response is given (carriage return only), the loading function is terminated and the user is returned to the system selection level or build mode.

PATCH-LODX responds with a "?" indicating readiness to accept the first patch. The patch data must consist of a one to six digit octal address, delimited by a blank, which in turn must be followed by any number of 1- to 12-digit octal fields (the patch data), separated by commas. Successive question marks are issued to obtain patches until receipt of only a carriage return, "*", or "D". A carriage return causes reissuance of the "PATCH, SAVE OR RUN?" query, while an "*" or "D" causes control to be passed to the loaded program.

PATCH filedescr - The specified file is used as the patch source. The format of the file is exactly the same as a series of patches entered from the keyboard. A patch file created by the text editor may also contain the "*" or "D" indicator to enable program execution. If an end-of-file or any error is encountered, the "PATCH, SAVE OR RUN?" query is reissued.

The PATCH function of LODX, LODT, and LODS accepts patches that are formatted for the $PATCH section of startup. A blank terminates the patch data and allows comments and/or module catalog names to be included on the line containing the patch(es); e.g.

1	8	16	32	73
243	OCTAL	5600004	TZE 5,IC	.TSACC

SAVE - The loaded program is stored back on the H* file from which it was obtained. Note that the file now contains a single program element, regardless of how many elements were initially present.

SAVE filedescr - If the specified file exists, LODX saves the loaded program in H* format on this file. If insufficient space exists, an attempt is made to grow the file or, if the file does not exist, it is created for the user at this time. The trace package is not included on the saved file when LODT or LODS has been specified.

SAVE filedescr;progname - The loaded program is appended as an additional element on the specified file with a name corresponding to "progname". The name must consist of 1-6 alphabetic and/or numeric characters (period or dash is also permitted).

RUN - The loaded program is entered for execution at the entry address specified in the control block of the H* file.

RUN nnnnnn - Same as above, except an alternate octal entry address, nnnnnn, is desired by the user.

Examples

1) LODX SAVEHSTR

2) LODX USERA/EXEC/GAME

Purpose

The LOGOFF command terminates a user session, prints usage statistics and disconnects the terminal.

Format

LOGO[FF]

Description

The LOGOFF command is an alternative to the BYE command used to terminate the current session.

Purpose

The LUCID command is used instead of the TAPE command to read paper tape for non-ASCII paper tape input.

Format

LUCI[D]

Discussion

The input is stored on a temporary file (TAP*) file as unaltered eight-bit ASCII character codes. The TAP* file is left open (unedited in the user's AFT). When a pause greater than one second stops the tape read, the system returns to the subsystem selection level. This command does not function when data communication is via a Low Speed Line Adapter (LSLA) or an Asynchronous Communications Base (ACB) on a DATANET 355/6600 Front-End Network Processor. In the EDITOR subsystem, this command takes the form #LUCID.

TAP* can be copied to a permanent file by the user via the PERM or CPY command, if desired.

Purpose

The **MAIL** subsystem command options consist of five keywords which are interpreted as "act on" words, augmented with two specialized options (asterisk and parenthesis). These action words are CREATE, DELETE, LIST, LISTL, or LISTD. Any subset of the keywords string may be used to identify the command.

Format

The total command syntax list is made up of the following nine groups.

1) MAIL

MAIL followed immediately by a carriage return implies that the user wishes all messages sent to that Userid to be dsiplayed at the terminal. If no messages exist, the statement "NO MAIL AT THIS TIME" will be displayed. This syntax additionally sets a flag which subsequently allows the message "YOU HAVE MAIL" to be sent to that user-ID by TSS LOGON. In effect, the use of this syntax functions as a "receive mail" feature that gives the user the option to elect whether or not to see the message during future logons.

2) MAIL*

Functionally similar to MAIL (or) except that the "NO MAIL AT THIS TIME" is suppressed if no messages exist on the file. This syntax also resets the above-mentioned flag so that the "YOU HAVE MAIL" message is not sent to that user-ID during subsequent logons.

3) MAIL CREATE (or any subset of create)

Directs the subsystem to create a "MAIL.BOX" file for the user. The initial size of this file is two llinks.

NOTE: This file will be increased in size by the subsystem as needed for larger messages up to the maximum limit determined by the space available under the SMC.

4) MAIL DELETE (or any subset)

Deletes all messages sent to the user's MAILBOX.

MAIL DELETE UID,UID...UID

Deletes all messages sent to the MAIL.BOX by the user-ID or user-IDs specified.

5) MAIL LIST (or any subset of list)

Directs the subsystem to list the header information for each message on the user's MAIL.BOX file without displaying the actual message content. The header information consists of the total number of messages sent, the user-IDs that sent them, the number of characters in each message, and the date and time each message was sent.

MAIL LIST UID

Displays the header data along with all messages sent to the mailbox from the user-ID specified, along with the header data.

6) MAIL LISTL (no subset permitted)

Causes the header data to be displayed showing how many messages have been sent to the requesting user. Also displayed are the date, time and user-ID of the latest messages sent (i.e., those with the current date).

MAIL LISTL UID,UID...UID

Displays all messages sent by the user-IDs specified that have a date in their header equivalent to the current date.

7) MAIL LISTD (no subset permitted)

Lists all header data of messages sent to the user and concurrently deletes them.

MAIL LISTD UID,UID...UID

Lists the header data and the text of all messages sent by the user-IDs specified and concurrently deletes them.

8) MAIL UID TEXT

Directs the MAIL subsystem to send the message text as entered to the user-ID specified. (note: the message text is not in *SRC unless using format #9.) The user sending the mail will be informed if the receiving user-ID did in fact receive the message. The message initiator will receive the message "USERID NOTIFIED", where USERID is the user-ID of the receiving user.

MAIL UID,UID...UID TEXT

Causes the text entered to be sent to the user-IDs specified.

MAIL UID

Directs the subsystem to prompt the user for the message input. The user will be prompted until a null line is encountered (response is carriage return only). Upon receiving the null line, the entered message will be sent to the specified user-ID.

MAIL UID,UID...UID

Similar in function to the above syntax, however the text is sent to all the user-IDs specified.

9) A. MAIL =CAT/FILENAME
Prompts the user for input and sends the text to the user-IDs specified on the file (FILENAME).

B. MAIL =CAT/FILENAME TEXT
Sends the entered text to all the user-IDs specified on the file (FILENAME).

C. MAIL =CAT/FILENAME-1 =CAT/FILENAME-2
The message contents on file FILENAME-2 are sent to the user-IDs listed on file FILENAME-1.

D. MAIL UID =CAT/FILENAME
Notifies the subsystem that the user wishes to send the message on the file FILENAME to the user-ID specified.

E. MAIL UID,U D...UID =CAT/FILENAME
Serves the same function as the previous command except the message is sent to the specified user-IDs.

Discussion

When a file is to be used by the MAIL subsystem it must be in standard system format and contain ASCII records. The user of the contents of the file is determined by whether the FILENAME is positionally the "first" or "second" option on the line. In items 9A and 9B, the FILENAME specification is the "first" option and denotes the file contents as user-IDs. In items 9D and 9E, the file content is determined to be text, rather than user-IDs, because the file specification is the "second" option in the syntax. In item 9C both FILENAME options are present, so FILENAME-1 (first) contains user-IDs and FILENAME-2 (second) contains text.

It should be noted that any command of the form UID,UID...UID can specify from one to N user-IDs. Note also the use of the term subset. As stated previously, this implies the use of C,CR,CRE,CREA, etc. for the CREATE command.

Purpose

The MAST command invokes the MAST subsystem for the user whose ID is MASTER. The MASTER user identity (default MASTER) is established in the TSS communication region. The MAST command and its subfunctions are privileged and as such are unavailable to normal users.

Format

```
MAST <functions>

<function>        ::= DONE
                      |MESS
                      |MONI[TOR]
                      |MSOF
                      |MUPD[ATE]
                      |PATC[H]
                      |PRIO[RITY]
                      |PSWD
                      |SPEC
                      |SMCL
                      |SNAP
                      |SSPA[TCH]
                      |STAT[US]
                      |TALK
                      |TCAL[L]
                      |UPDA[ATE]
                      |WHOS[ON]
                      |PROF[ILE]
                      |AFT
                      |VERB
                      |PEEK
```

Description

DONE - Used to exit the MASTER subsystem and control is returned to the build mode.

If the master user does not wish to continue with the use of the MAST subsystem, he gives the response DONE to the selection request. The MAST subsystem is then dispensed with and control is returned to the build mode level.

MESS - The MESS function permits a message to be issued to all currently active terminals and all those terminals that subsequently become active. Up to 68 characters, including line-feed and carriage-return characters, can be written in one or two lines. The request symbol is ?. Two lines of input or a carriage return immediately after the ? indicates the end of the message. A call to MESS and a carriage return after the first ? serves to clear out a message that is no longer needed.

```
?TSS WILL DUMP FILES AT 2250 ON 10/17/76.
?
```

The message, issued to all active users, is prefixed by the time of day that the MESS function was exercised.

MONITOR - The MOI ITOR function allows the master user to select a terminal and receive all input/output to and from that other terminal concurrently until the monitored user disconnects, goes into TAPE mode, or the master user presses the BREAK key.

In response to LINE NUMBER ?, the user must give an octal number designating the user line id. If the specified User Status Table (UST) is active (and not occupied by the master user himself), all input/output occurring at the corresponding user's terminal is received at the master's terminal also.

The master user may determine the line id corresponding to a user-ID by using the WHOSON function.

In attempting to monitor a terminal, the master user may receive one of the following error messages, which indicate the condition preventing the monitoring function. After each error message, the function question is asked.

MONITOR IN PROGRESS

TERMINAL NOT TTY

ILLEGAL CHAR

CANNOT MONITOR SELF

USER NOT CONNECTED

USER IN TAPE MODE

MSOF - The MSOF function sets .TMSON to a value greater than zero and, in effect, prevents anyone from logging on as master user.

MUPDATE - All users, or specific users, can have their total resources initialized to a given dollar amount and/or the resources used to date initialized to a given dollar amount (e.g., 0). Any previous amounts are ignored. An SMC list report is output with the old values.

Entry to this function is similar to that of SMCL; i.e., requests for initial values and the user entry list. The following is an example (user responses are underlined):

INITIALIZE TOTAL RESOURCES? n or carriage return

INITIALIZE RESOURCES USED TO DATE? n or carriage return

Where n = dollar amount (integers)

?user-ID1 carriage return for all users or list specific users ($\leq$25)
?user-ID2
?carriage return

UPDATE SYSTEM MASTER CATALOG ENTRIES

RESOURCES= amount entered above

RESOURCES USED TO DATE= amount entered above

USER ID	PRIORITY	PASSWORD	MAX# BLOCKS	BLOCKS USED	RESRCS	RESRCS USED

The use of MUPb (MULTIPLE-UPDATE, NO-HEADER) suppresses the header information.

Error messages are the same as for SMCL.

PATCH - The PATCH function allows the user to make modifications and changes in the TSS Executive coding and in associated data tables. The patches are in force until TSS is reloaded. Following issuance of a line feed and question mark, the user must reply with a location (1-6 digits), followed by one blank and one or more data fields (1-12 digits), separated by commas. The general form, including both the request and response, is:

```
?location data,data,...,data
```

Because of the variable format, a line may contain a variable number of data fields, but each line must first specify the location. A blank, other than the one separating location and data, terminates the line. A response of carriage return or **** indicates that the function is complete.

All location and data values must be specified in octal. All locations are expressed relative to zero, in accordance with the memory map produced by the General Loader.

```
?70000 1
?325 1437,01400000,767774000001,1
?14372 1235007,755012,14402710000
?(carriage return)
```

If illegal characters are typed, the following error message occurs:

ILLEGAL FORMAT--RETYPE

PRIORITY - The PRIORITY function gives the master user control over the use of the restricted options LODX, LODS, TALK, and CARDIN. Using the PRIORITY function, the master user may:

1. Grant permission to all or specific users to use the LODX, LODS, TALK and CARDIN options.

2. Set a maximum initial urgency for a user using CARDIN.

3. Withdraw permission given any user to use the LODX, LODS, TALK and CARDIN options.

4. List all or specific users who have permission to use the restricted options and list the batch job urgency of users using CARDIN.

   NOTE: The master user has LODX and LODS permission implicitly.

Input is requested, and the master user can respond with one of the following subfunctions:

ADD system options .../name options ...

DEL (or DELETE) system options.../name options...

LIST name options

(carriage return) done with PRIO function.

The system options are LODX, LODS, TALK, CARD (or CARDIN), or CARD(XX), where XX is a number (1-40) indicating the maximum urgency of the user's batch job in CARDIN. The urgency designation must be enclosed in parentheses and immediately follow the CARD option. More than one option/name option set may be specified in one directive, with each set separated from every other set by commas. For example:

?ADD option/name,option/name...

A continuation line is accepted when a line ends with a comma or a slash, or following the command ADD, DELETE, or LIST.

The name options are user-IDs, separated by slashes, or the word ALL*, which indicates that the command applies to all users in the System Master Catalog.

The ADD directive is used to grant permission to use LODX, LODS, TALK and CARDIN to all or specified users (listed by name in the option field) and to set urgency for CARDIN users. For example, to add LODX users ABC, LMN, and XYZ:

?ADD LODX/ABC/LMN/XYZ

To add these users for CARDIN with an urgency of (10):

?ADD CARDIN(10)/ABC/LMN/XYZ

To add users for both subsystems:

?ADD CARDIN(10),LODX/ABC/LMN/XYZ

To add all users as CARDIN users with no special urgency:

```
?ADD CARDIN/ALL*
```

To use more than one subsystem user set in the same directive:

```
?ADD LODX/ABC/XYZ,CARDIN/ABC/XYZ/LMN,LODS/ABC/LMN
```

The DEL (or DELETE) directive is used to withdraw permission to use LODX, LODS, TALK and/or CARDIN from specified users (listed by name in the option field). It is formatted the same as the ADD except that the urgency is not allowed. For example, to delete users ABC and XYZ from LODX and CARDIN, user LMN from CARDIN only and user XYZ from TALK:

```
?DEL LODX/ABC/XYZ,CARDI/ABC/XYZ/LMN,TALK/XYZ
```

The LIST directive is used to list all users who have permission to use CARDIN, LODX, LODS, TALK and to list the batch job urgency of CARDIN users. For example:

```
?LIST ALL*
```

or

```
?LIST ABC/LMN/QRS
```

where only those users that are specified are listed.

A user who has been named in an ADD directive can be added again. This feature can be used to change the urgency level of a CARDIN user. Adding a user to one option does not delete that user from another option.

When the command has been executed, the system sends a SUCCESSFUL! message to the master user. If an error occurs in processing user-IDs, the system sends the master user an appropriate error message. There are four groups of error messages, as follows:

Group 1 - Syntactical error detected in command string; the master user is asked to reissue his command:

```
ILLEGAL COMMAND - START OVER
INPUT REQUIRED - START OVER
ILLEGAL URGENCY - START OVER
ILLEGAL SYSTEM - START OVER
ILLEGAL USE OF ALL* - START OVER
ILLEGAL DELIMITER - START OVER
```

Group 2 - Name table overflow error:

```
TOO MANY NAMES, WILL PROCESS 25
```

PSWD - The PSWD function allows the user to determine the octal encrypted form of a password for the purpose of preparing patch cards for the startup deck. In response to FUNCTION?, the user enters PSWD, which results in an eight-character strikeover mask being issued at the terminal. One to eight characters must be entered at this time. The pa sword is encrypted and displayed at the terminal as two octal words, 12 digits each. For example:

```
USER ID - MASTER
PASSWORD
XXXXXXXX
PASSWORD
XXXXXXXX
*MAST
FUNCTION? PSWD
XXXXXXXX (user enters desired password on strikeover mask)
012345670123  456701234567
```

The password string may include any combination of uppercase or lowercase alphabetics, numerics, special characters, and nonprinting characters such as BELL, TAB, etc. (but not including SPACE). If fewer than eight characters are entered, trailing blanks are supplied to compensate for the difference. Note that nonprinting characters offer an obvious advantage for security purposes.

SPEC - The SPEC function permits a message to be issued to a specific terminal. Up to 68 characters, including line feed and carriage return, can be entered in one or two lines. The first prompt will be a ?. The desired terminal identification may be entered in one of three ways.

1. 1- to 12-character user-ID.
2. =nnnn Station id in four octal characters.
3. =xx Station id in two BCD characters.

Following the station identification, a ? is issued for the message text. If the user-ID form is used, the message is sent to all terminals that are signed on with that user-ID and all subsequent terminals that sign on with that user-ID. A response of the carriage return to the first ? will cancel all specific messages. Entering a new specific message will cancel any previous specific message. The message issued to a specific user is prefixed by the time of day that the SPEC function was exercised.

SMCL - The SMCL (System-Master-Catalog List) function allows the master user to list the contents of the System Master Catalog at his terminal. A request (?) for input is issued. The master user may respond with a carriage return, in which case the entire System Master Catalog (SMC) is listed. The master user also has the choice of specifying which entries in the SMC are to be listed by supplying a list of IDs. Up to 25 names (IDs) may be entered. A request (?) for input is issued repeatedly until a carriage return response is made, after which the designated SMC entries are listed in the order specified.

More than one ID may be placed on a line, each separated from one another by a comma or a slash.

Each SMC entry occupies one line in the listing; the format of the line is as follows:

USER ID	PRIORITY	PASSWORD	MAX. # BLOCKS	BLOCKS USED	RESRCS	RESRCS USED

The resource figures are in round dollars.

The use of SMCØ (SYSTEM-MASTER-CATALOG LIST, NO-HEADER) suppresses the header information.

Error messages that may be issued are as follows:

TOO MANY CHARACTERS IN NAME. START OVER.

TOO MANY NAMES, WILL LIST 25.

CANNOT IDENTIFY

SNAP - The SNAP function allows the master user to select and display areas of memory at a terminal. At the request ?, the master user supplies an address for a one-word snap, or for a multiple-word snap, an initial and final address, separated by a dash, or an initial address and the number of words to be snapped separated by a comma. Any locations within the memory area assigned to the TSS may be snapped. A carriage return or **** indicates that the function is complete. Locations are expressed relative to the TSS LAL, in accordance with the memory map produced by the General Loader. When requesting a snapshot, an addend may be specified with the initial address. A response of 1430+7,3 would result in words 1437, 1440, and 1441 being snapped. A * may be used to indicate the last location snapped. A sequence 21700,5 followed by a *+1,3 would result in locations 21700 through 21705 being snapped and then locations 21706, 21707, and 21710 being snapped.

All addresses and number of words are assumed to be octal. The output is double-spaced, with a location followed by four words of data on each line in octal.

```
?2100
(snapshot)
?1437,14
(snapshot)
?26037-26045
(snapshot)
?(carriage return)
```

The following error message may occur:

ILLEGAL FORMAT -- RETYPE

SSPATCH - The SSPATCH function, utilized for subsystem patching, enables the master user to modify subsystems. The name of the subsystem to be patched must be the first response. The name must be four characters long; blanks are required to fill out shorter names (e.g., NEWb̸). Then locations and data, in the same format as for PATCH, are given. All locations and addresses must be in octal; addresses are expressed relative to zero, in accordance with the memory map of the subsystem produced by the General Loader.

STATUS - The STATUS function provides the master user with a detailed status report of TSS operation. This report provides the accumulated statistics on TSS operation from startup to the time of the status request.

The first section of the report gives an accounting of number of available lines, total logon time, a percentage of available line time used (busy time), number of users, number of disconnects, number of subsystem starts, system interactions, etc. The following is a sample of this part of the report:

```
                            TSS STATUS

     01/12/77    3.810      TO       01/13/77      10.924

 50 LINES          10.616  HRS LOGON TIME            19% BUSY

     79 USERS                0 REJECTS             65 BREAKS
     17 MAX USERS           52 DISCONNECTS        796 SS STARTS
     14 CUR USERS           65 TERMINATES         746 SS KILLS

   1769 ALARMS(WAIT I/O)             0 ALARMS(NO USERS)
    650 SWAPOUTS                  4226 K SWAP OUT
    434 KEY I/O SWAPOUTS            24 SNUMB(CARDIN JOBS)
     60 K CORE

   %INTERACTIONS/        %INTERACTIONS/          %INTERACTIONS/
   ELAPSED TIME(SEC)      PROGRAM SIZE              # FILE I-O

 PERCENT  INTERVAL       PERCENT  K(SIZE)        PERCENT   #I-O
    87       2             57       2              94       10
     8       5              4       5               2       25
     2       9             23       8               1       50
     .       .              .       .               .        .
     .       .              .       .               .        .
     .       .              .       .               .        .
```

The derail usage section of the status report lists all derails with a count of the number of times each derail is used. This is formatted as in the following sample.

```
                           ***DRL USAGE***

     DRL     COUNT     DRL    COUNT     DRL    COUNT     DRL    COUNT
      1     173846     18       45      35       0       52        8
      2      69194     19     1763      36       0       53       10
      3      17993     20     6124      37    5573       54     3916
      .        .        .        .       .       .        .        .
```

In the subsystem usage section of the report, the usage of each subsystem is described in terms of processor time (in seconds), file I/O time, key I/O count (number of characters/8) issued to the users' terminals during the executions of each subsystem, number of times each subsystem was called at build mode level, command level, or automatically (as with BSED), and the number of times each subsystem was loaded for execution. This data is formatted as follows:

SUBSYSTEM USAGE

NAME	PROC TIME	# FILE I/O	# KEY I/O	# CALLS	# EXEC
BASI	.000	0	0	20	0
ACCE	.740	0	4032	7	7
CARD	.000	0	0	48	0
EDIT	.000	0	0	11	0
EDTX	50.924	2283	5648	18	18
BYE	.000	0	0	11	0

The subsystem usage information is totalled at the bottom of the subsystem usage table. The final entry in the report is the total TSS processor time. (The total processor time for the subsystems is less than the total subsystem processor time, because file building does not require a subsystem and, therefore, is not reflected in the total subsystem processor time figure.)

TALK - The TALK function permits the master user to converse with the operator at the console. The master user must indicate if a response is expected. A maximum of three lines is accepted as a message or a response.

RESPONSE ? yes, no, or carriage return (done)

MESSAGE ? (followed by line feed and carriage return)

(message)

? (message or carriage return)

? (message or carriage return)

Error messages that may occur are as follows:

ILLEGAL FORMAT -- RETYPE
COFFEE BREAK -- TRY AGAIN (console operator has failed to respond)

At the completion of a response and message, if requested the cycle will be repeated starting with the RESPONSE question.

TCALL - The TCALL function permits the master user to disconnect any terminal. The terminal to be disconnected can be specified by the user-ID or by its station code. All terminals that have the designated user-ID or station code will be disconnected. Deferred users can be terminated either via their user-ID or by a station code of 2020, which will terminate all such users. The response to the prompting ? may be a one-to-twelve character user-ID, an octal station code (=nnnn), an alphanumeric station code (=xx), or *ALL to disconnect all terminals.

Error message that may occur is as follows:

TCALL BUSY TRY LATER

UPDATE - The UPDATE function allows for updating of the System Master Catalog (SMC) from the master user's terminal. The subsystem prints a question mark (?), which is the request for input. The master user can then request one of the following subfunctions:

?ADD id,res,pass,lnks	Add a user to SMC
?DEL id	Delete a user from SMC and release his file space
?INS id,res	Increment a user's resources in SMC
?REP id,pass	Replace a user's password in SMC
?LKS id,lnks	Change the number of links assigned to user
?END or (carriage return)	Done with UPDATE function

where: id is the 1-12 character user identification.
res is a decimal number representing even-dollar resources.
pass is the 1-12 character password.
lnks is the number of links to be allocated to the user, ($\leq$21,845 links) or
UNLIMITED to allow an unlimited number of links to be allocated.

One blank must follow the subfunction name, but no embedded blanks may occur in the data.

Error messages that may occur are as follows:

ILLEGAL FORMAT -- RETYPE
NON-UNIQUE ID
CANT IDENTIFY - RETYPE
STATUS ERROR n (where n is the status character)
NEW LINK SIZE LINKS USED - RETYPE
#LINKS > 21,845 -- RETYPE

In the Delete (DEL) subfunction, special error handling may be required. The system will ask:

TOO MANY LEVELS/DELETE NOT COMPLETE
ERROR IN FILE RELEASE
DELETE STILL WANTED?

If the response is YES, the SMC is released before the system asks for more input. If an error occurred in releasing files under a User Master Catalog (UMC), it should not affect release of the SMC entry if this release is still desired. However, the file space of a user deleted by a YES response is not returned to the system until a total system initialization is performed.

WHOSON - The WHOSON function generates (on the master user's terminal) a list of the current time sharing users and their respective line ids by which they are connected. The line id is used to assist in the MONITOR function.

AFT - The AFT command allows the MASTER user visibility into another user's available file table entries to possibly track file conflict problems. The system will prompt the MASTER user with a question mark (?) to obtain the four digit line identifier of the user whose AFT is to be displayed.

EXAMPLE:

```
*MAST AFT (CR)
?2310(CR)
SY**  *SRC    TEMP
?2240(CR)
SY**  *CFP
(CR)   DONE
```

The function is coded in such a manner that a null response will terminate the AFT scanning process; otherwise, another user's line identifier may be entered. The routine uses the DRL T.EXEC facility to locate the desired user in the UST chain by comparing the user line identifiers in .LBUF with the line identifier entered by the MASTER user. Once the correct user has been located, the subsystem enters master mode to use the pointer in .LFILE to obtain the file names from the user's chained PATs. After the file names have been gathered, the DRL T.EXEC routine is terminated to allow main level processing to display the user's AFT contents.

PEEK - The PEEK command allows the MASTER user to display the contents of memory locations beyond the bounds of TSS. In concept, the PEEK command is similar to the SNAP command except that the SNAP command displays memory locations within TSS. The use of the PEEK command is shown below:

SYNTAX:

```
PEEK ADDRESS,NUMBERWORDS,PROGRAMNUMBER
```

In the PEEK command syntax, the fields will be octal numbers if present. It is possible to explicitly null the number of words to imply a PEEK of one word. If a program number is not given, the address of the PEEK will be assumed as an absolute address. If a program number is given, the PEEK address will be considered as an offset to the specific program's base address. The maximum number of words that may be displayed at any one time is 1024. The implementation of the PEEK command is based on a DRL T.CMOV call and all rules and restrictions associated with the DRL T.CMOV are applicable to the PEEK command. Various samples of the PEEK command are shown below:

EXAMPLE:

```
*MAST PEEK(CR)
?700(CR)          absolute location 700(8), 1 word
?700,5(CR)        absolute location 700(8), 5 words
?100,3,5(CR)      100(8) relative to program 5, 3 words
?700,,5(CR)       700(8) relative to program 5, 1 word
```

PROF - The PROF (PROFILE) command gives the MASTER user the capability to receive a summarization of any user's activities. The display produced by the PROF command contains information about the user and the status of the session. Included in the report are the following items:

USERID for the user
Station identifier
UST address
Subsystem size
Last derail location
Last derail number (type)
Contents of UST flag words.LFLAG/.LFLG2.LFLG3
Subsystem BAR
Contents of UST switch words.LSWTH/.LSWT2
Stack level indication
Stack entry name

The PROF command will prompt the MASTER user for a station identifier to be used as the unique user-ID. The four digit station identifier is used to scan the USTs to locate the profiled user.

EXAMPLE:

```
*MAST PROF(CR)
?2210(CR)       Station B8 / 2210
```

The routine uses the DRL T.EXEC interface to scan the USTs to locate a particular user for which the display is desired. Once the user has been found, the profile information is gathered from the UST for later display and formatting by a main level routine.

VERB - The VERB capability within the MAST subsystem allows the MASTER user to place console verb input queue entries in the core allocator's input queue for processing. The interface to provide console verb capabilities is a one-way interface; no provision has been made to capture the console output generated by the verbs entered by the MASTER user. The VERB processing will prompt the MASTER user for a console verb by issuing a question mark.

EXAMPLE:

```
*MAST VERB(CR)
?LSTAL(CR)
```

Any valid console verb and its associated option may be entered in response to the prompt. The verb and option will be entered in the same manner as would have been given at the operator's console. As an example, the MASTER user might choose to type "LIST LIMITS" in response to the prompt. The VERB routine uses the DRL T.EXEC interface to perform the user/console functions. Once a verb and optional parameters have been entered and syntactically checked, a GECALL is issued to load the console verb handler (.MPOP7) to peruse the verb list for content verification. If the MASTER user has entered a valid verb, then a MME .EMM is executed to issue a .CALL to .MPOPM, EP3 (common queuer mechanism) to place the queue entry for the console verb into MPOPM's input queue. Of particular note in the VERB processing is the transition from master mode coding within a floatable subsystem to slave relocatable coding. The DRL T.EXEC routines must be floatable and the return using the TSS instruction will not function correctly since the floatable TSS implies the use of a "master mode" IC. Therefore, the TSS back to the subsystem had to be coded using an address register in addition to the IC modification to ensure the return was to the correct location. The address register must be loaded with the complement of the TSS LAL to execute properly. An sample of the coding required to accomplish the return is as follows:

EXAMPLE:

```
      (Within the floatable routine of the DRL T.EXEC)
        MME     .EMM
        LDQ     P1,$
        STQ     P2,$
        SWDX    0,5,AR1
        TSS     LBL,$,AR1
LBL     NULL
```

Although an assembly error results from the above construct, the execution of the instruction functions correctly.

Discussion

When the master user dials the TSS, he gives the MASTER user-ID (default MASTER) and then gives his initial master password, which has been assigned by the computer-installation authority. At this point, the logon routine recognizes the master user as a special user and requests a second password. The response is a second master password, also set by the installation. The master user is given the customary two chances to type his user-ID and first password correctly; the second password must be typed correctly the first time or there is an immediate disconnect.

The master user then selects the subsystem MAST. The message FUNCTION? is issued, and the user responds with his selection. Any or all of the MAST functions may require a password. A password flag is provided in MAST that allows the site to mark those functions that it wants to require a password. If a function is designated to require a password, requesting that function will produce an eight-character strikeover mask that is an implicit request to enter the password. An encrypted form of this password is assembled into the MAST subsystem and is set by the installation.

The first function to be performed can be included on the MAST command line; e.g.,

*MAST STAT

Purpose

The MDQ command invokes the Management Data Query System.

Format

MDQ

Discussion

Refer to the Management DataQuery System (MDS) Basic User's Guide for further information.

Purpose

The NEW command causes an empty current file to be created for use.

Format

NEW [P<filesdecr>|P#<filedescr>]

Discussion

At LOGON the system sets up a temporary current file, named *SRC, on which new files are built or on which selected old files are copied. The user can make the current file permanent by issuing the SAVE/RESA commands.

When the user selects the NEWP format to create a current file, the named permanent file is created and opened with an alternate name of *SRC. The permanent file remains the current file until another OLD or NEW command is entered.

The NEWP# format functions similarly to the NEWP form. The named file remains the current file until either an OLD, OLDP, OLDP#, NEWP or NEWP# command is issued.

Examples

NEW

NEW FILENAME

NEWP FILES

NEWP# FILESAVE

Purpose

The NEWUSER command initiates a new LOGON and reports the user's system usage in the following terms:

- o dollars used during the current "LOGON" session
- o dollars used to date during this billing period and the percent of the monthly allotment that amount represents
- o the storage LLINKS in use, the total LLINKS allocated to the user-ID and the percent of those LLINKS that are in use.

Format

NEWU[SER][<charge number>]

Discussion

Before the usage report is printed, the Available File Table (AFT) is scanned for user's temporary files. A message is issued as to the number of temporary files, then the user is queried as to their disposition. Each temporary file name is printed followed by a question mark. The user may respond as follows:

1) carriage return - the file is not to be saved.

2) NONE - this file and all of the succeeding temporary files are not to be saved.

3) SAVE [<filedesc>] - the temporary file is to be saved in the permanent file specified by <filedesc>. If the permanent file does not already exist, it is created with general READ permission.

Examples

1) NEWU

2) NEWU CHG03 (retains the current user-ID for the session and begins a new accumulation of charges under account number CHG03).

NFORM
NOPARITY/PARITY

Purpose

The NFORM command allows the user to inhibit the transmission (from the Time Sharing Executive) of the form-feed character after the prompt for a page request.

Format

NFORM

Discussion

NFORM operation (as opposed to FORM) on keyboard/display devices causes each succeeding page to overwrite the preceding page. FORM operation clears the screen before displaying the next page. The NFORM command is not valid when the terminal is under the effect of the PAGE command.

Purpose

The NOPARITY/PARITY command turns off or turns back on the adding of parity on user's output by the system.

Format

{NOPA[RITY]|PARI[TY]}

Discussion

The system normally adds even parity to the user's output record before sending it to the terminal. The typical user must have parity added to the output record, as most terminals check for it.

However, there are special applications which require data without parity added. If the user wants a paper tape, cassette tape, or other storage media generated without parity, the NOPARITY command can be used to turn off the adding of parity. The command PARITY should be given before continuing with regular output to the terminal.

Examples

1) NOPARITY (turn off the adding of parity)

2) PARITY (turn back on the adding of parity)

Purpose

The OLD command copies the specified ASCII time sharing file, files or file segments onto the current file.

Format

```
OLD [ <file-ref>[ { ;|:| } <file-ref>]...]

<file-ref>        ::= { *|<filedesc>}[(<line-range>)]
<line-range>      ::= <begin-line>-<end-line>
                  |  <begin-line>-
                  |  -<end-line>
<begin-line>      ::= <line>
<end-line>        ::= <line>
<line>            ::= a one- to eight digit decimal number
```

Discussion

If a semicolon is used to separate the file names as in Examples 3, 5 and 6, the contents of the files specified are joined in the order listed and copied onto the current file. The files need not have lines with steadily increasing line numbers. No automatic sorting or resequencing of the lines takes place.

If a colon is used to separate the file names as in Example 8, and the files have lines with steadily increasing line numbers, the contents of the files specified are weaved or merged according to line number and copied onto the current file. If lines with the same line number appear in more than one file, they are all kept and merged according to the order of the file names listed.

If a colon is used to separate the file names and the files do not have lines with steadily increasing line numbers, an error occurs.

If a pound sign is used to separate the file names, the action taken is the same as with a colon except that when lines with the same line number appear in more than one file, only the last such line is retained.

If a combination of semicolons, colons and pound signs is used to separate the file names as in Example 9, the concatenating and merging is performed in the order the files are listed (left to right). If however, after concatenation, the resulting file does not have lines with steadily increasing line numbers, and merging is the next function, an error occurs.

If the file list is too long for one line, the OLD subsystem will request more input when a semi-colon, colon, or pound sign is the last nonblank character before the carriage return.

The following formats describe options associated with the OLD command.

1. OLD filedescr (permissions and altname applicable)

   File filedescr becomes the current file.

2. OLD filedescr(i,j) (permissions and altname applicable)

   Lines i through j of file filedescr become the current file. Filedescr must be a line-numbered file.

3. OLD f-1(i,j);...;f-n(i,j) (permissions and altname applicable)

   The n files or file segments are adjoined in the order listed and become the current file, where f is a filedescr. Adjoining of BASIC files should be done with caution (sequence numbers are also statement numbers). The asterisk designating the contents of the current file (or segment thereof) may appear as a filedescr anywhere in the file list.

   Note that these files or segments are concatenated on the current file and resequencing may be required for satisfactory operation in line-number dependent systems. Sorting or resequencing is not automatic.

4. OLD f-1(i,j):...:f-n(i,j) (permissions and altnames applicable)

   The n files or file segments are merged by line numbers, and become the current file, where f is a filedescr (colon-separated). If duplicately numbered statements appear in two or more files, each such statement appears in the order specified by the file list. If it is desired to retain only the last duplicately numbered statement, the colons may be replaced by pound signs (#). The asterisk designating the contents of the current file (or segment thereof) may appear as a filedescr anywhere in the file list.

5. OLD f-1(i,j);f-2(i,j):f-3(i,j);...:f-n(i,j) (permissions and altname applicable)

   A combination of forms (3) and (4). Concatenation or merging is performed in the order (from left to right) indicated by the file list.

   If the file list is too long for one line, the OLD subsystem will request more input when a delimiter is the last nonblank character before the carriage return.

6. OLDP filedescr (permissions applicable)

   The specified permanent file is accessed with an alternate name of *SRC and becomes the current file. This file is the user's current file until another form of the OLD or NEW command is given. The contents of the file will always be checked or verified for Time Sharing System format.

7. OLDP# filedescr (permissions applicable)

   Execution is the same as for the OLDP command, except that this file remains the user's current file until logoff, or until another OLDP, OLDP#, NEWP, or NEWP# command is given. The normal OLD or NEW commands use this file (i.e., the file specified by OLDP# or NEWP#) as the current file. OLDP# can be cancelled by REMOVE *SRC.

The OLDN subsystem is called in when the command OLD, NEW or LIB (normal forms) are given by the user. If a NEWP or OLDP command was issued and then one of the normal forms was typed in, OLDN will deaccess the permanent *SRC file and assign a new temporary *SRC file to the user. The permanent file remains in the user's catalog until he releases it.

If a NEWP# or OLDP# command was issued and then one of the normal forms was typed in, OLDN will retain the permanent file as *SRC. If a NEWP or OLDP was typed in instead of the normal form, the permanent *SRC will be deaccessed, and a new permanent file with the alternate name *SRC will be created and/or accessed.

If a NEWP# or OLDP# command was issued and then followed by another NEWP# or OLDP# command, the OLDN subsystem will deaccess the present *SRC file and then create and/or access the newly specified *SRC file.

Merging and concatenation are not allowed with OLDP, NEWP, OLDP#, and NEWP#.

Examples

1) OLD (prompts for the file name)

2) OLD DATA74
   (the contents of file DATA74 replace the prior contents of the current file)

3) OLD MAIN;SUB1;SUB2
   (the contents of files MAIN, SUB1 and SUB2 are concatenated in the order listed and replace the prior contents of the current file)

4) OLD *(10-200)
   (lines 10 through 200 of the current file replace the prior contents of the current file, i.e., any lines outside the range 10 through 200 are deleted)

5) OLD WEIGHTS(100-500);VOLUME(100-500)
   (lines 100 through 500 of file VOLUME are appended to lines 100 through 500 of file WEIGHTS and they replace the prior contents of the current file)

6) OLD /PROJECT1/S.SUB1;/PROJECT1/S.SUB2
   (concatenates in the order listed the files S.SUB1 and S.SUB2 that are in subcatalog PROJECT1, and replaces the prior contents of the current file)

7) OLD NAILS/BRADS,R
(the contents of the file BRADS in the user-ID NAILS replace the prior contents of the current file)

8) OLD DATA73:DATA74
(the contents of file DATA73 are merged with the contents of file DATA74 and they replace the prior contents of the current file)

9) OLD PROG1:FIXES;SUB1;SUB2#ALTERS
(merges the contents of the files PROG1 and FIXES, concatenates with the result the contents of files SUB1 and SUB2, and merges the contents of the file ALTERS with the result)

Purpose

The PAGE command allows a device (e.g. 7800 Series VIP) to display output in a scrolled fashion. Although primarily intended for high speed devices, the PAGE command is applicable to all devices.

Format

```
     PAGE {-OFF|<n>[-LF<000>...][-RL<000>...][-FF<000>...][-PE<000>...]}

    -LF   ::= set line feed type character, add one to line count
    -RL   ::= set reverse line feed character, subtract one from line count
    -FF   ::= set form feed character, top page without page erase
    -PE   ::= Page Erase
    <n>   ::= set the number of lines per page >0
   -OFF   ::= turn off existing page mode
  <000>   ::= specifies a seven-bit ASCII character (000-177)
```

Discussion

The PAGE command is implicitly invoked when the <n> parameter setting is used. When no values are specified for the optional fields, the following default values apply: -LF=012, -FF=014, and -PE=024. If any of the optional values are used, no default values apply. Up to eight characters in total may be indicated for any combination of the four types. As an example, the following PAGE command sets the number of lines per page to 24 and causes the line-feed (012) and carriage return (015) characters to increment the line counter.

Example

```
PAGE 24 -LF 012 015 -FF 014
```

Purpose

The PASSWORD command allows any properly validated user to change the current System Master Catalog (SMC) password.

Format

PASS[WORD] subsystem prompts for old and new passwords

Discussion

The process involved in changing an SMC password requires the user to go through three steps: the user is asked

1) for his current SMC password for validation purposes.

2) for the new SMC password.

3) to reenter the new password for validity checking.

Example

```
*PASS
ENTER OLD PASSWORD
XXXXXXXXXXXX
ENTER NEW PASSWORD
XXXXXXXXXXXX
REENTER NEW PASSWORD
XXXXXXXXXXXX
```

Purpose

The PERM command copies a temporary file to a permanent file and removes the temporary filename f·om the Available File Table (AFT).

Format

```
PERM[ <temp-filename>[;<perm-file>]]

<temp-filename>    ::= a 1- to 8-character name of a temporary file
<perm-file>        ::= <filedesc>
```

Discussion

If the permanent file does not already exist, it is created with general READ permission.

Examples

```
1)   PERM
                    (prompts for the file names)

2)   PERM TEMP1;PERM1

3)   PERM TEMP2;TEMP2
                    (effectively makes TEMP2 permanent)

4)   PERM DAILY;JUN75$PW
                    (creates a permanent file JUN75 with a password PW)

5)   PERM TEMP3;/SUBCAT/PERM3
```

PRINT

Purpose

The PRINT command reformats and then prints on the terminal the specified ASCII time sharing file, files or file segments.

Format

```
PRIN[T][ <file-list>|<line-list>]

<file-list>    ::= <file-ref>[;<file-ref>]...
<file-ref>     ::= {*|<filedesc>}[(<line-list>)]
<line-list>    ::= <line-ref>[,<line-ref>]...
<line-ref>     ::= <line>|<line-range>
<line-range>   ::= <begin-line>-<end-line>
                |  <begin-line>-
                |  -<end-line>
<line>         ::= a 1- to 8-digit decimal number
```

Discussion

A question/answer sequence to acquire the reformatting options is initiated by this command unless the first file name contains reformatting information in the first line.

Examples

1) PRINT

(reformats and prints the current file)

2) PRINT *

(same as PRINT)

3) PRINT 10-100

(reformats and prints lines 10 through 100 of the current file)

4) PRIN 100,200,300-350

(reformats and prints lines 100, 200 and 300 through 350 of the current file)

5) PRINT PROG5

(reformats and prints file PROG5)

6) PRIN DATA74(1000-1999)

(reformats and prints lines 1000 through 1999 of file DATA74)

7) PRIN DATA75(10,100,200)

(reformats and prints lines 10, 20 and 200 of file DATA75)

8) PRINT MONDAY;TUESDAY

9) PRINT DAILY;*;/REPORT/MONTHLY

10) PRINT USERID/FILE,R

Purpose

The PURGE command releases the specified file(s) from the file system and overwrites the released file space. (Refer also to the RELEASE and ERASE commands.)

Format

PURG[E] [<filedescr->;<filedescr-2>;...;<filedescr-n>]

Examples

PURG FILEA (remove FILEA from the file system and overwrite it)

PTON
PTOF

Purpose

The PTON command enables subsystem output to be routed to the print page adapter attached to a VIP770[1] terminal. The PTOF command disables the printer mode of operation.

Format

PTON|PTOF

Discussion

PTON and PTOF are commands for VIP type terminals so that output can be directed to an associated printer.

Examples

```
*OLD TEST
*PTON
*LIST  (output goes to printer - not displayed)
*PTOF
*

*NEW
*PTON
*CATA  (catalog output goes to  printer - not displayed)
*PTOF
*
```

Purpose

The READ command causes a specified tape cassette to be read.

Format

READ TAPE[n]

[n] ::= 1 or 2, the default value is 1.

Discussion

This command can only be used on VIP 7700 Series VIP (device 13, 14 or 15 octal).

Purpose

The permanent file designated in the command input is created and/or accessed as the input collector file for the user.

Format

RECO[VERY] <filedescr>

Discussion

The RECOVERY subsystem dumps data currently on the temporary input collector file to the current file and creates and/or accesses a permanent file specified in the command.

Examples

1)
```
*RECO FIL1$ABC
RECOVERY NOW IN EFFECT
*
```

2)
```
*BASIC OLDP FIL2
*RECO FIL3
RECOVERY NOW IN EFFECT
*
```

3)
```
*EDIT NEWP .SRC
*#REC FIL4
RECOVERY NOW IN EFFECT
*
```

4)
```
*BASIC NEWP SOURCE
*RECO FIL6$CAB
RECOVERY NOW IN EFFECT
*10 PRINT "THIS IS LINE #10"
*20 PRINT "THIS IS LINE #20"
*30 PRINT "THIS IS LINE #30"
*40 PRINT "THIS IS LINE #40"
*50 PRINT "THIS IS LINE #50"
*60 PRINT "THIS IS LINE #60"
*70 PRINT "THIS IS LINE #70.
```

Assume that at this point the computer system disconnects. After logging back in, the user will do the following to recover his last input lines.

```
*BASIC OLDP SOURCE
*ROLL FIL6$CAB
FIRST AND LAST LINES OR SAVED DATA ARE:
10 PRINT "THIS IS LINE #10"
70 PRINT "THIS IS LINE #70"
RECOVERY NOW IN EFFECT
*
```

When the system is restarted after the disconnect, the user calls in the RECOVERY subsystem by issuing the ROLLBACK command. The RECOVERY subsystem will access FIL6 and sort and merge the data onto the current working file. When the RECOVERY NOW IN EFFECT message is issued, the user is ready to type into an empty collector file.

Purpose

The RELEASE command releases the specified file(s) from the file system.

Format

RELE[ASE][<filedesc>[;<filedesc>]...]

Discussion

The file content is not overwritten or destroyed before the file space is released to the file system for reallocation to other files. The user has no means for requesting that same file space again, which means that, effectively, the content is lost. But, if the user is really concerned about a possible breach of security, the content should be destroyed before releasing the file space.

Refer to the ERASE command to overwrite the file content.

Examples

1) RELEASE (prompts for the file name)

2) RELEASE DATA 74 (releases the file DATA74 from the file system)

3) RELE TYPE;MAKE;MODEL
(releases the files TYPE, MAKE and MODEL from the file system)

4) RELE /UTILITY JCL
(releases the file JCL in the subcatalog UTILITY from the file system)

Purpose

The REMOVE command removes file names from the Available File Table (AFT).

Format

```
REMO[VE][ <aft-ref>]

   <aft-ref>  ::= PERMFILES
                | TEMPFILES
                | CLEARFILES
                | <filename>[;<filename>]...
   <filename> ::= a 1- to 8-character name
```

Discussion

Refer to the description of the Available File Table in Section II of this manual for more details.

Examples

1) REMO
   (prompts for the file names to be removed)

2) REMOVE TEMPFILES
   (removes all temporary file names, except *SRC and SY**, from the AFT)

3) REMO PERMFILES
   (removes all permanent file names from the AFT)

4) REMO CLEARFILES
   (removes both temporary and permanent file names, except *SRC and SY**. from the AFT)

5) REMOVE SOURCE
   (removes the file name SOURCE from the AFT)

6) REMO ASMBLJCL;RUNJCL
   (removes the file names ASMBLJCL and RUNJCL from the AFT)

7) REMO *SRC
   (removes the current file, *SRC, from the AFT)

Purpose

The RESAVE command saves the contents of the current file on an existing permanent or temporary file(s), replacing their prior contents.

Format

RESA[VE][<filedesc>];<filedesc>]...]

Discussion

If the file name is qualified, (i.e., it is preceded by a slash or a subcatalog reference) and the file name is in the AFT, it is assumed that the user may not want to use the file name that is in the AFT.

Examples

1) RESAVE
   (prompts for the file name)

2) RESA WKFILE
   (copies the contents of the current file onto the existing file WKFILE replacing the prior contents of WKFILE)

3) RESA PROG1;EXTRA
   (copies the contents of the current file onto the existing files PROG1 and EXTRA replacing the prior contents of those files)

4) RESA /LACQUER/GLOSS
   (copies the contents of the current file onto the existing file GLOSS in the subcatalog LACQUER replacing the prior contents of that file)

Purpose

The RESEQUENCE command resequences the line numbers of the current file (*SRC) when under control of BASIC.

Format

RESE[QUENCE]|RESEX|RESE#[n,m,x-y]

n ::= value of beginning line number
m ::= value to increment by
x-y ::= beginning and ending line numbers of a partial resequencing

Discussion

When resequencing, or performing a partial resequence, it is possible to produce files with line numbers out of order. This may be caused by incorrect parameters on partial resequence or when new line numbers exceed eight digits (in non BASIC files). When line numbers are too large, a warning is given. In either case, recovery may be made by resequencing the total file using a smaller beginning line number or a smaller increment.

Examples

1) RESEQUENCE

The line numbers of the current file are resequenced. The resequencing begins with line number 10 and continues in increments of 10. If BASIC is the selected subsystem, the file is resequenced and statement number references in the program are modified correspondingly (GOTO, GOSUB, IF, ON, PRINT USING).

2) RESEQUENCE _n_,_m_,_x-y_

The line numbers of the current file are resequenced and modifications made according to the subsystem selection. The resequencing begins with line number _n_ and continues in increments of _m_.

_x_ and _y_ are specified only if partial resequencing is desired. _x_ gives the starting point and _y_ the ending point of resequencing, inclusive. A null _x_ field (i.e., -_y_) specifies from beginning of file to line _y_, and a null _y_ field (i.e., _x_-) specifies from line _x_ to the end of file.

In general, any blanks preceding a line number are stripped off. Unnumbered lines are accepted, except under the BASIC subsystem, and will have line numbers added, as implied or specified in the command. Care should be taken in resequencing concatenated BASIC files as line numbers are also statement numbers, and statement references, after resequencing, may become invalid.

3) RESEX n,m

Line numbers are inserted at the beginning of each line in the current file, regardless of whether or not line numbers already exist. The numbering begins wit' n and increments by m, or optionally, begins with 10 and increments ›y 10, if n,m are not specified. If the first character of the existing line is a numeric, a blank is inserted following the generated line number. If the first character of the existing line is not numeric, no blank is inserted.

4) RESE# n,m

Line numbers are inserted at the beginning of each line in the current file, even if line numbers already exist. This numbering begins with n and increments by m, or optionally begins with 10 and increments by 10 if n, m are not specified. If the first character of the existing line is a numeric, a pound sign (#) is inserted following the generated line number. If the first character of the existing line is not numeric, the pound sign is not inserted.

Purpose

The REW command rewinds a specified tape cassette.

Format

REW TAPE [n]

[n] ::= 1 or 2, default value is 1.

Discussion

The command can only be used on 7700 Series VIP (device 13, 14 or 15 octal).

Purpose

The ROLLBACK command accesses a permanent file with read and write permissions and it becomes the input collector file for the user.

Format

ROLL[BACK] <filedescr>

<filedescr> ::= a permanent file

Discussion

Any data lines previously collected on the file will be merged with the current file and the first and last such lines of recovered data will be shown to the user. #ROLL is the format used in the Text Editor and the recovered data is appended to the current file, instead of merged. The ROLL command is used in conjunction with the RECOVERY command.

Examples

See RECOVERY command.

Purpose

The RUNOFF command calls the RUNOFF subsystem into use to process an EDITOR prepared text file for formatted output.

Format

RUNO[FF]

Discussion

The RUNOFF subsystem allows the user to reformat a text file and then print a listing of the reformatted file, and/or copy the reformatted file to another file. The format is determined by special control words in the file. Some of the formatting features available are paragraphs justified to a right margin, top and bottom margins, titles centered on the page, and pages numbered consecutively.

Refer to a later section for details of the RUNOFF subsystem.

Examples

RUNOFF

Purpose

The SABT command calls the SABT (Scan ABort) subsystem into use to peruse the contents of the file \BRT. The ABRT file contains the memory image of a subsystem fault condition.

Format

SABT

Description

When a fault occurs in a subsystem which does not handle such faults or a DRL abort is executed, and the user has a file named "ABRT" opened, the aborted subsystem (or program) is copied into the file. By means of SABT the user may scan the ABRT file by snapping portions of it at the terminal. Also using SABT the user can initiate a batch print of the ABRT file contents.

Discussion

The SABT (Scan ABorT) subsystem provides special purpose scanning functions. When a fault occurs that is not handled by a subsystem or a DRL ABORT is executed, the aborted subsystem is copied onto a file called ABRT. The file ABRT must exist previously and must be in the Available File Table at the time of the abort. The user can then scan the ABRT file by snapping portions of it at the terminal.

Examples

```
1)   *SABT
     OFFSET?W       (print a listing of the ABRT file contents)

2)   *SABT
     OFFSET?W AB (makes output available to remote station AB)

3)   *SABT
     OFFSET?
     ?100     (snap loc 100)
     ?100,4   (snap locs 100 through 103)
     ?100-110 (snap locs 100 through 110)
```

Purpose

The SAVE command saves the contents of the current file on a new permanent file or files.

Format

```
SAVE[ <filedesc>[;<filedesc>]...]
```

Discussion

The permanent file is created with general READ permission.

Examples

1) SAVE
(prompts for the file name)

2) SAVE DATA74
(creates a file called DATA74 and copies the contents of the current file onto the file DATA74)

3) SAVE S.PROG;EXTRA
(creates two files called S.PROG and EXTRA and copies the contents of the current file onto those files)

4) SAVE /SOURCE/PROG01
(creates a file called PROG01in the existing subcatalog SOURCE and copies the contents of the current file onto the file PROG01)

Purpose

The SCAN command calls the SCAN subsystem into use for perusal of the generated output of batch jobs.

Format

SCAN[<filedesc>]

Discussion

The SCAN subsystem is a batch-output scanner. It allows highly selective listing of any type of batch output or any standard-system-format BCD file.

SCAN responds with the question FORM?
To this respond:

```
FORT  -- for FORTRAN compilations
GMAP  -- for GMAP assembles
LOAD  -- for loader output
COBOL -- for COBOL compilations
USER  -- for all others
```

For the answers GMAP, LOAD, COBOL, SCAN responds with the number of errors that occurred.

For the answer USER -- SCAN responds with the question code? -- the normal answer to this is a carriage return (if any characters are typed in then subsequent FIND and PRINT command will ignore any lines that do not begin with these characters). Any characters typed are called the line code.

Next the question EDIT? is asked

```
     A response of YES -- for multiple-blank suppression
                   NO  -- for printing multiple blanks as is
                   A null response is the same as a no response
                   A * suppresses line numbers
```

SCAN should return with a question mark, at which point you may enter any of the following scan verbs:

F or FIND[/<STRING>/][;N]

The slash (/) represents any desired delimiter chosen by the user. The string is a pattern of characters to be searched for. N (any integer) is used to find the Nth occurrence of a string.

The FIND verb is used to locate text in a report, and to position the search pointer in a forward direction. The F verb operates only from where the pointer is to the end of the file. The FIND verb positions an implied pointer to the Nth line containing the literal string (beginning with the line currently pointed to). If N is not given, 1 is assumed. If no literal string is given, all lines are assumed to match.

The FIND verb accepts all standard Honeywell text EDITOR forms.

Examples:

```
       FIND /FORMAT/        will find the next line containing the word
or     F /FORMAT/           "format" or the current line
       FIND ;1                    will do nothing
       F ;2                       will move the pointer ahead one line
       FIND /X/;4             will find the fourth line with an "X" in it
       FS
```

P, PS, or PRINT[/<STRING>/][;N;*]

N is the number of lines to print. If N is the character string all, then the rest of the lines from the current line to the end are printed. If the string * is used, then all lines containing a string matching the <STRING> will be printed. If no arguments are given, only the current line is printed.

Print allows the user to inspect the next N lines of text in a report, or the next N lines which match a specified character string. The search pointer does not move. Lines are printed with a scan line number which can be used with the line verb.

The PRINT verb accepts all standard Honeywell text EDITOR forms.

```
Examples:  P 5
           P /ACTY/;*
           P /FORMAT/;3
```

S or SPACE [N]

Spaces the pointer ahead N lines. If N is not specified, the pointer will advance one line. If the user attempts to position the file beyond its end, the file will be positioned at the beginning, with a warning message issued at the terminal (EOF).

B or BACK [N]

Spaces the pointer back N lines. If N is not given the pointer is moved back to line 1.

Examples:	B	(return to top of report)
	BACK 1	(back up one line)
	B 25	(back up 25 lines)

LINE [N]

As each line is listed, an automatically generated line number will be typed with it. The LINE verb repositions the pointer to the specified line number, N. (The line number used need not have been printed prior to being referred to.)

E or ERROR [N]

Requests a list of the next N error printouts of the form corresponding to the output form in question. The absence of N implies all such messages.

U or UNDE

This command (no argument) is used while scanning GMAP assembles to list all undefined symbols.

FLAG [X]

Lists all lines of a GMAP assembly having the error flag specified by X (X equals A, U, M, O, etc). The absence of a specific error tag implies that the user wishes a list of all flagged instructions.

LOAD

Prints out an abbreviated load map. Only primary SYMDEFs are listed, and library routines are omitted.

C or CODE ABCDE

Employed with the user format to change the line code. The argument ABCDE is a 1- to 5-character code of BCD characters. A null argument "turns off" line codes. That is, all lines codes are accepted until the code verb is used to resume with a valid line code.

EDIT

Returns the user to the EDIT? level.

DONE
D

Returns the user to the prior level depending on how the SCAN was invoked.

LIST [N]

LIST is synonymous with PRINT in all respects.

REM [TEXT]

The REM verb provides a means of placing a remarks line on the terminal session log, it if is being taken on a hardcopy terminal.

BATCH

The BATCH verb asks STATION CODE? The user replies AB or simply a carriage return, where AB is the station code of a remote-batch terminal.

The system then asks, $ IDENT?, to which the user replies the variable field of the user's BATCH $ IDENT card.

The BATCH verb initiates a Bulk Media Conversion (BMC) job which transfer the entire contents of the file to remote printer AB. If the station-code reply is null, the output will be printed at the host.

BYE

Terminates user's current session with the time sharing system.

Examples

1) SCAN
(prompts for the file name)

2) SCAN BCDFILE

Purpose

The SEND command cancels the effect of a previous HOLD command and send the last message withheld.

Format

SEND

Discussion

Refer to the HOLD command for more details.

Example

SEND

Purpose

The **SEQUENCE** command inserts line numbers at the beginning of each line of the current file. If the first character of an existing line is numeric, a number sign (#) is inserted following the generated line number. Otherwise, no such number sign is inserted.

Format

```
SEQU[ENCE]#[ [<initial>][,<increment>]]

   <initial>  ::= <line>
   <increment>::= <line>
   <line>     ::= 1- to 8-digit decimal number
```

Description

<initial> the 1- to 8-digit number to be used as the first line number. The default value is 10.

<increment> the increment between line numbers. The default value is 10.

Discussion

If there is a number sign immediately after the line number, it is treated as part of the line number.

Examples

1) SEQU# 10,10
   (inserts line numbers using 10 as the beginning line number and 10 as the increment)

2) SEQU#
   (same as Example 1)

3) SEQUENCE# 100,5
   (inserts line numbers using 100 as the beginning line number and 5 as the increment)

4) SEQU# 1000
   (inserts line numbers using 1000 as the first line number and 10 as the increment)

5) SEQU# ,5
   (inserts line numbers using 10 as the first line number and 5 as the increment)

Purpose

The SEQUENCEX command inserts line numbers at the beginning of each line of the current file. If the first character of an existing line is numeric, a blank is inserted following the generated line number. Otherwise, no such blank is inserted.

Format

```
SEQU[ENCE]X[ [<initial>][,<increment>]]

   <initial>    ::= <line>
   <increment>  ::= <line>
   <line>       ::= a one- to eight-digit decimal number
```

Description

<initial> the 1- to 8-digit number to be used as the first line number. The default value is 10.

<increment> the increment between line numbers. The default value is 10.

Examples

1) SEQUX 10,10
   (inserts line numbers using 10 as the beginning line number and 10 as the increment)

2) SEQUX
   (same as Example 1)

3) SEQUENCEX 100,5
   (inserts line numbers using 100 as the beginning line number and 5 as the increment)

4) SEQUX 1000
   (inserts line numbers using 1000 as the first line number and 10 as the increment)

5) SEQUX ,5
   (inserts line numbers using 10 as the first line number and 5 as the increment)

Purpose

The SMCL command is a request to display the user's System Master Catalog.

Format

```
SMCL
```

Discussion

The System Master Catalog contains the user-ID, password, maximum number of llinks permitted for saving files, current number of llinks in use, maximum dollar resources permitted, resources used, and certain permissions assigned by site personnel. SMCL will not display the password.

Example

```
*SMCL
   USER ID-JDOE
   MAX LLINKS - UNLMTD
   LLINKS USED - 908
   MAX RESOURCES - $10000
   RESOURCES USED - $1484.71
   PERMISSIONS-X S T C(35)
```

The permissions are:

Symbol	Name	Meaning
X	LODX	The user is permitted to load and execute bound programs residing on an H* or Q* file
S	LODS	The user is permitted to use the LODS subsystem for debugging resident TSS software.
T	TALK	The user is permitted use of the CARDIN TALK option
C	CARDIN	The user is permitted to run batch jobs. The suffixed parenthesized quantity represents the highest job urgency that is allowed.

Purpose

The SORT command invokes the SORT subsystem for sorting files.

Format

SORT [infile][=otfile][:F1,F2...Fn/S1,S2...Sn<*><99>

[infile] ::= filedescr

[=otfile] ::= filedescr

Fi ::= sort field descriptors

Si ::= sort sequence designators

* ::= an optional parameter specifying deletion of duplicate records

<99> ::= an optional parameter representing the dominate record size for the input records.

Discussion

In addition to the normal representation of the current file (*SRC) as "*", the following abbreviations in SORT command syntax show different assumptions that may be made regarding the input and output files for the SORT:

```
SORT  *=*:options (current file input/current file output)
SORT  IN:options  (file IN input/current file output)
SORT  =OT:options (current file input/file OT output)
SORT  :options    (current file input/current file output)
```

The SORT subsystem uses the user-supplied arguments to construct a DRL TASK interface to the assembler in a batch activity. The source images provided to the assembler are built using the sort macro expansions normally used in a batch sort submission. Once the DRL TASK for the assembly returns, another DRL TASK is initiated to execute the object code in a loader phase. The bulk of the SORT subsystem processing deals with the correct syntax analysis of the user input(s) so the DRL TASK submissions will perform correctly; no dump facility exists for DRL TASK job abort analysis.

The user-supplied field and sequence parameters for the sort are provided in a compatible manner to the batch sort options. For example, the user input to the sort for an ASCII file using the first four characters as the key might appear as ---SORT IN:A4/A1. Note the A4 field parameter is constructed from the permissible ranges (A,B,C,W) and values (numeric representation of a four-character field). The sequence parameter similarly corresponds to its batch equivalent (i.e., ascending sort sequence using the first field parameter as the key (A1)). When the length of the field or sequence parameters is known to be large, the user may elect to be prompted for input rather than constrain the length of the input strings to the length of the current line.

Examples

1) *SORT ABC/CAT/FILEIN=ABC/SOURCE/OTFILE(CR)
FIELD? A3,W2,A4,W4(CR)
SEQUENCE? D3,A1,D2(CR)

When the user knows the dominant size of the sort input records is different than the default size of 40 words, it may be useful and more efficient to supply a dominant record size as part of the command syntax. The dominant record sized field is a one or two digit number contained in parenthesis that occurs on the input line following the sequence parameters or the optional duplicate record deletion character (*).

2)	*SORT IN=OT:A12/A1(4)(CR)	dominant record size is 4 words Major sort field (A12) is 12 ASCII characters in length. Sort of field is ascending (A1).
3)	*SORT IN=OT:A12/A1*(4)	Same as previous example, except duplicate records are deleted in the sort output.

Purpose

The STATUS command lists the names of the files that are in the Available File Table (AFT) and/or reports the user's status as to the station ID of the terminal, the processor time used, the number of file I/Os completed, and the number of characters output to the terminal.

Format

STAT[US][F|FILES]

Examples

1) STATUS

(lists the names of the user's files in the AFT and reports the user's status)

2) STATUSF

(lists the names of the user's files in the AFT)

3) STATF

(same as STATUSF)

Purpose

The STRIP command strips the line numbers from the current file.

Format

STRI[P][B|X|#]

Discussion

The STRIP command removes leading blanks and line numbers from columns 1-8 of the current file. The STRIPB command strips trailing blanks from the current file. If a text line in a file consists of only a line number, STRIPB appends a single blank to the line number. STRIPX strips the line numbers from the current file and also the trailing blank if the next character of the line is numeric. The STRIP# command strips the line numbers and trailing pound signs from the current file.

Purpose

The SYSTEM command cancels any previously made system selection and causes the named selection to ιualify the meaning of any subsequence RUN or RESEQUENCE commands.

Format

SYST[EM] <name>

<name> ::= name of any time sharing subsystem

Discussion

If the name of a subsystem is not specified, SYSTEM only cancels the previous system selection and subsequent RUN or RESEQUENCE commands result in unpredictable situations.

Example

```
*BASIC
*SYSTEM EDITOR
```

Purpose

The TAPE command initiates the reading of paper tape.

Format

TAPE
#TAP[E] EDITOR subsystem

Discussion

In order to supply file-building input from paper tape, the user gives the command TAPE (#TAP if the subsystem is Text Editor). The subsystem responds with READY. If the tape reader is ready, it will be turned on automatically. If it is not ready, the user should position the tape in the tape reader and start the device. Input is terminated when an X-OFF character (DC3 on some terminals) is read by the paper tape reader, or the tape is stopped and the user types X-OFF (or DC3).

The tape may be prepared offline from the keyboard, or it may be the result of previous output punched by the paper tape unit. If prepared offline, it should include carriage returns to terminate each line, just as if entering data online, plus explicit line feeds to obtain legibility on the terminal printer during preparation and transmission. The carriage return and line feed must be followed by two rubout characters for terminal timing considerations.

Command language may not be included on the tape. The input should be preceded by several rubout characters and terminated by an X-OFF (or DC3) followed by several rubout characters. Neither the X-OFF (or DC3) nor the rubout characters will appear in the file.

As with keyboard input, a maximum of 160 characters is permitted per line of paper tape input. Excessive lines are truncated at 160 characters, with the remaining data placed in the next line. A maximum of two disk links (7680 words) of paper tape input will be collected during a single input procedure, except in LUCID mode, which has a limit of six links. All excess data will be lost.

In order to supply file building input from non-ASCII paper tape (unaltered eight-bit codes), the user gives the command LUCID instead of TAPE. The system reads in the tape and stores the data on a file without editing or parity modifications. The system does not delete or act on any characters in the data stream, such as DEL, X-OFF, DC3, CR, etc. The input is terminated when a pause of over one second occurs in the data transmission. Termination does not require an X-OFF (or DC3) character, as does normal paper tape input via a Front-End Network Processor.

NOTE: LUCID cannot be used if data communication is via a Low-Speed Line Adapter (LSLA) or an Asynchronous Communication Base (ACB) on a DATANET 355/6600 Front-End Network Processor.

During paper tape input via a Front-End Network Processor, the paper tape input will stop when an error message is to be sent to the terminal. At any point during the operation of the Time Sharing System and at a time when the user must supply keyboard input, a previously prepared paper tape in special format may be used to simulate a sequence of responses, one line at a time. The system need not be in build mode and direct (i.e., conversational) responses, file building input, and/or commands may be entered.

This feature allows the preparation of a paper tape for input to the Time Sharing System and/or subsystem(s) prior to connection with the system and allows terminal operation without supervision during the connection. The paper tape input may be for a specific subsystem or production program execution only, or may include anything from logon through logoff procedures. Obviously such a tape must be error-free.

The required format for each input line is as follows:

```
data string (up to 80 characters)
carriage return
X-OFF (or DC3)
RUBOUT (may be multiple, but one is minimum requirement)
```

Character-delete control characters may be used. Line-delete controls must be used as follows:

```
data string (to be deleted)
(line-delete control) character
X-OFF (or DC3)
RUBOUT (one is minimum)
corrected data string
carriage return
X-OFF (or DC3)
RUBOUT
```

Parity errors encountered during paper tape input may cause the terminal to be disconnected.

It is suggested that extraneous line feeds not be included in the tape. If, however, the user desires line feeds for terminal printer legibility, they should be either between the data string and carriage return, or one line feed immediately following X-OFF (or DC3).

To initiate automatic paper tape input, the user should position the tape and start the reader any time keyboard input is required.

The terminal is automatically disconnected if no input is received within 10 minutes of the request for such input, whether via paper tape or keyboard.

Purpose

The TEMP command creates a temporary file or files.

Format

```
TEMP  (prompts for inputs)

<temp-filename>    :- a one- to eight-character name of a temporary file
<mode>             ::= L[INKED]
                    |  R[ANDOM]
<SIZE>             ::= a one to three digit decimal number
```

Description

<temp-filename> — the 1- to 8-character name of the file, optionally enclosed in quotes. If <filename> could be misconstrued as <size>, <size-unit> or <mode>, it must be enclosed in quotes as shown in Example 6.

<size> — the initial size of the temporary file. The default is 1 LINK, which is equivalent to 12 LLINKS. (BLOCKS and LLINKS are different names for the same unit.) if the size is specified in LLINKS, the size is rounded to an integral number of LINKS. The maximum size allowed is 25 LINKS, or 300 LLINKS.

<mode> — is the type of file being created. The default is sequential.

Discussion

A temporary file can be used in other commands such as RESAVE, ASCBCD, etc. and can be read from and written to in a program.

The PERM command can be used to make a temporary file permanent.

At sign-off, the system will ask what is to be done with the temporary files. Refer to the BYTE, NEWUSER, or LOGON command for a description of the responses that can be given.

Purpose

The TEX command invokes the TEX subsystem

Format

TEX

Discussion

Any file building, editing or commands will be under control of the TEX subsystem. See the TEX manual (DF72) for further details.

Purpose

The TSAR command invokes the Master Time Sharing Activity Report subsystem. The use of the TSAR subsystem is restricted to the MASTER use only.

Format

TSAR

Discussion

Since TSAR is a master subsystem, the user-ID must be MASTER and requires two passwords.

At entry to the subsystem, the user is shown a list of the available displays from which to select the numeric equivalent and give a time (a whole number) to observe the display.

```
TIME SHARING SYSTEM MONITOR

1.    ALLOCATION
2.    ACCUMULATED STATUS
3.    SUBSYSTEM USAGE
4.    DONE

SELECT A DISPLAY AND GIVE A TIME(MINUTES)

* (selection,time)
```

A short routine validates the subsystem selection (screen display terminal), and displays the selection list. The time, or implied time if none is given, is stored, a form feed is issued (clear screen), and the routine is called to process the selection. If a time value of zero is specified, output is produced for the appropriate selection and the user is requested to make another selection.

Examples

The following is an example of Display 1:

```
            TIME SHARING SYSTEM MONITOR

            1.    ALLOCATION
            2.    ACCUMULATED STATUS
            3.    SUBSYSTEM USAGE
            4.    DONE

  SELECT A DISPLAY AND GIVE A TIME(MINUTES)
   *1,1
     SWAP AREA    35K            (21 to 56)
     (1111111222 0000000000  0000000000 00000)


                      ALLOCATION
       IN CORE          SWAP FILE         BUILD MODE
        MASTER        K SMITH           SER
        K APM585      K BISNET          BISNET
                      K 543             MOE
                      K MPCTND
                      K ASESA
                      K SQASD
                      K LJM
                      K AEP
                      K PASSAGE
                      K PROJECT2
                      K JONES
                        MASTER
```

The following is an example of Display 2:

```
          TIME SHARING SYSTEM MONITOR

          1.  ALLOCATION
          2.  ACCUMULATED STATUS
          3.  SUBSYSTEM USAGE
          4.  DONE

  SELECT A DISPLAY AND GIVE A TIME(MINUTES)
  *2,1
                 STATUS TSS
      02/04/72   8.932  TO  02.14.72  13.253

   35 LINES     7.077 HRS LOGON     99 HELLOS
   98 USERS         0 URGENT USERS   0 REJECTS
   14 MAX USERS   12 CUR USTS       86 DISCONS
   10 CUR USERS                     97 TERMS

   31965 ALARMS(WAIT I/O)      1 ALARMS(NO USERS)
    1559 SWAP OUTS         11906 K SWAP OUT
     557 KEY I/O SWAPS        58 SNUMBS
      38 K CORE              117 BREAKS

    0 SYSTEM ERRORS

     7 MME GEMORE  0 REFUSALS  31 LKS OBTAINED
     0 MME GERELS               0 LKS RELEASED

       TOTAL TSS PROCESSOR TIME   .291 HOURS
```

The following is an example of Display 3:

```
          TIME SHARING SYSTEM MONITOR

          1.  ALLOCATION
          2.  ACCUMULATED STATUS
          3.  SUBSYSTEM USAGE
          4.  DONE

   SELECT A DISPLAY AND GIVE A TIME(MINUTES)
*3,1
SYSTEM NAMES?
```

```
                  SUBSYSTEM USAGE
NAME      PROC TIME    FILE I/O     KEY I/O    #CALLS
           SECONDS    #CONNECTS      #CHAR
basy       113.508       1417        62344       56
edtx        82.233       3544        28104       59
cdin        57.531       1784         1456       37
ascb        27.807       1034          856       10
runy        16.080        532         8216       15
oldn        12.727       2716         4344      197
bsed        11.892        719         1096      137
jout        11.063       1754        11976       10
list         7.961        445        58096      123
rese         7.473        342          144       19
save         6.146       1318         3456      127
runo         5.699        178        25616       10
prin         5.239        134        28384       24
run          5.238        119         3136       14
lodx         5.106        228        14568       34
cata         2.418          0         9384       20
new          2.401          0        15320      104
```

The following is an example of Display 4:

```
          TIME SHARING SYSTEM MONITOR

          1.   ALLOCATION
          2.   ACCUMULATED STATUS
          3.   SUBSYSTEM USAGE
          4.   DONE

SELECT A DISPLAY AND GIVE A TIME(MINUTES)
*4
*BYE
**COST:  $    0.22 TO DATE:  $  316.53= 32%
**ON AT 11.583 - OFF AT 11.589 ON  04/25/75
```

Purpose

The TSDA command invokes the Time Sharing Dump Analysis subsystem.

Format

TSDA (subsystem prompts for verbs)

TSDA verbs

```
aft-display files in available file tables
alc-display TSSA allocator work cells
apb-display last all points bulletin
atr-display status of allocator (TSSL) trace
clr-clear all offsets presently in effect
cmp-display core map cells and memory use
coq-display jobs queued waiting memory
cur-display "current" user data
cwk-display UST mgmt work-hole list
def-display info about deferred processes
dic-display info about last drl from TSSK
dmo-display offset in dump by TSS module
dof-display all offsets
drl-display drl locs from map in TSSK
duo-display offset in dump for UST
emm-display .emm trace table from TSSM
end-terminate dump analysis
err-display TSSA error cells+TSSF trace
esq-display executive service queue
gat-display closed gates on system
his-display history register values
hld-place terminal into idle mode
ici-display ic+i value from SSA
inf-display meanings for TSSA/ust/ssa cells
ioq-display nonzero I/O queues used by TSS
lal-display TSS base address
lvl-set different levels of offset
mem-display memory map for whole system
pgd-display program descriptor information
prq-display jobs queued up for processor
prt-format dump and send to printer
reg-display register values from SSA+spa
sdq-display sub-dispatch queue and header
smo-set offset in dump based on TSS module
snp-snap memory locations using any offset
sof-set offset in dump based on address
sos-display verb explanations
spa-display formatted slave prefix for TSS
ssa-display address of first SSA used by TSS
sts-display last console status message
sum-display summary about dump environment
swf-display information on swap files
tsm-display TSS specific message
tsq-display last TSS input queue entry
tst-display formatted trace (build by TSSB)
```

uad-display user-ID, lineid, and UST address
uof-set offset in dump to UST
uss-display user status table summary
ust-display user status table
voc-display vocabulary

Discussion

The time sharing dump analysis package provides a convenient and effective tool to be used in the online analysis of time sharing system failures. The examination of time sharing failures is performed from a terminal connected to the system. The use of the time sharing dump analysis package with dial-in capabilities to the front-end processor makes it possible for analysts at on-site or remote locations to examine time sharing failures. In addition to executive failure analysis the package can serve as a useful tool for site analysts or training personnel who have a need to study the structure of the TSS executive. An obvious use of the analysis package would be in the development of changes to the executive that may result in time sharing failures. It is possible to minimize the loss of valuable test time by quickly examining the image of the executive produced at failure time using the time sharing dump analysis package as opposed to waiting for listings to be generated.

Examples

```
FN??voc
sum cmp uss prq uof duo coq atr swf tst ioq aft sof pgd smo dmo drl snp reg ici
spa uad dof alc lal ssa dic sts def err apb tsq ust end prt voc sos inf clr lvl
his gat mem cur hld cwk sdq esq tsm emm
```

Another verb that can be useful to an analyst is the INF verb. The user can enter a logical name for a cell and receive an explanation of what the cell represents.

```
FN??inf .SSA
.SSA    -stack tally,ic+i stack
FN??inf  .TCUST
.TCUST -# USTs currently in use/ptr to first UST
FN??inf  .LBUF
.LBUF    -buffer addr / stat id
```

The SUM verb displays a short summary of the system environment for the terminal user. Included in the display are the system identifier, GCOS release identifier, date of the dump, time of the dump, configuration data, an interpretation of the location of the last transfer of control (using X1) might have been from, an indication of which SSA module was found in TSS's SSA at failure time, the .STATE and .STAT1 words for TSS from the SSA, and an interpretation of the .STATE word bits. The abbreviations used to represent the different bits in the program .STATE word are shown below:

abbreviations	bit	meaning
in-exec	0	in execution
in-que	1	program number in queue
ssa-load	2	system I/O in progress
gepr-ctl	3	gepr in control
gepr-nd	4	gepr needed
abt-ctl	5	abort in control
abt-req	6	abprt request received
swap-ctl	7	swap or move in control
swap-req	8	swap or move requested
pgm-ccal	9	program in courtesy call
ccall-wt	10	courtesy call waiting
pgm-dead	11	program dead-waiting size change
syot-I/O	12	sysout writing
reling	13	relinquished
rdblock	14	roadblocked
filler	15-17	
mme-.emm	18	master mode entry permitted
class-a	19	class-a priority
no-ccall	20	do not pay courtesy call
no-gepr	21	do not set abort bit
wrap-end	22	wrapup done
enable	23	enable request
gate-ssa	22	gated module busy
swap-ned	25	swap requested or in control
no-swap	26	do not swap or move
I/O-end	27	I/O complete since last link
alarmset	28	alarm set for program
swpdelay	29	swap or move delay
filler	30	
geidse	31	mme geidse done
no-proc3	32	processor-3 can't execute this
no-proc2	33	processor-2 can't execute this
no-proc1	34	processor-1 can't execute this
no-proc0	35	processor-0 can't execute this

**NOTE: With the six processor option it is unlikely that the last four strings will appear.

FN??sum dump on sys-sysa GCOSid-sr3 080677 17.027 1-pro 1-iom 384K last transfer of control via x1 from TSSJ (015046) last module in TSS SSA was .MALC9 TSS .STATE = 400400440400 TSS .STAT1=000000000000 in-exec pqm-ccal mme-.emm no-qepr I/O-end.

The core map verb, CMP, breaks out each entry of the TSS core map that is nonzero. The display first shows the three main control cells that administer the core map--1) .TACOR, 2) .TAMPT, and 3) .TAHOL. The breakout of the core map entries follows the display of the above cells. Any core map entry that represents a BTOS buffer entry is flagged as such.

```
FN??cmp
  *core map*
   .tacor            .tampt            .tahol            avail
054000162000      000413000415      000421000000          000070
next    #blks     #blks    SS         prev    UST        user-ID
entry   subsys    open     lal        entry   addr
000417  000       000      054        000     000000     000000000000
000000  002       100      060        002     051430     honeywell       BTOS
```

The UST summary display verb, USS, can display all current user status table summary entries or it can display a selected entry if the user enters a user-ID following the verb. Each UST summary entry shows the user-ID, UST address relative to TSS, the file list pointer, the subsystem size cell, the last derail cell, the program stack, the flag words, and the BTOS flag word. The breakout for the above entries consists of interpreting the last derail type code, marking the last stack entry, interpreting all valid stack entries, the interpreted bits from the flag words, and the interpreted BTOS flags. The abbreviations used to breakout the user summary display are shown below:

abbreviations for .LFLAG	bit	meaning
app-file-I/O	18	application file I/O process
rdblk-relinq	19	terminal I/O rdblk or exec relinquish
mast-ss-alc	20	master subsystem allocated
swapot-strt	21	swap out in progress
pgm-in-core	22	program in memory
swap in-strt	23	swap in in process
new-interact	24	new interaction
batch-job	25	non-TSS process in execution
apb-reqd	26	all points bulletin required
filler	27-29	
ccall-reqd	30	courtesy call required
pgm-swapped	31	program on swap file
pgm-alc-que	32	program in a allocator queue
fake-I/O	33	fake I/O
fswap-pend	34	force swap scheduled
brk/dis-recd	35	break or disconnect received

abbreviations for .LFLG2	bit	meaning
talk-mode-cf	4	user in talk mode of command file
drl-task-2	5	drl task phase indicator
logon	6	logon in progress
term-type	7	terminal type extension bit
command-file	8	command file processing in progress
vip-tape	9	7700 VIP cassette number
pat-size-flg	10	pat size indicator
reconn-xmit	11	reconnect mode with data transmit
hold-reconn	12	reconnect mode with line in hold
master-user	13	this is the master user
user-execute	14	user code executing
cmdl-reqd	15	command loader must be invoked
ids-user	16	ids user
pseudo-I/O	17	pseudo I/O in progress
ccall-entry	18	courtesy call entrance
buff-I/O-req	19	buffer flush required
vip-term	20	VIP terminal
line-switched	21	line switched to batch job
filler	22	
2wd-lineno	23	two-word line number
auto-blank	24	blank indicator in auto mode
auto-lineno	25	automatic line-numbered mode
fake-dump	26	fake dump command
end-take-flg	27	end of tape flag
filler	28	
ppt-xmit-err	29	error in transmit of ppt
ppt-file-err	30	error in file built from ppt
wait-end-ppt	31	wait for end of ppt
ppt-start	32	ppt in transmission
monitor-user	33	user being monitored
lastio-out	34	last remote I/O was output
lastio-ot/in	35	last remote I/O was input

abbreviations for .LPQF	bit	meaning
extra-buffer	18	extra buffers present in memory
data-xmit	19	data in transmission
ebm-refused	20	extra buffer request refused
buffers-full	21	all buffers full
buff-in-core	22	buffers in memory
buffio-disk	23	buffer I/O in progress from disk

primitive codes	primitive number
dummy	0
callp	1
exec	2
bin	3
popup	4
discon	5
xcall	6
system	7
ifalse	8
itrue	9
stfals	10
strue	11

```
FN??uss honeywell
**ust summary**

   user-ID       UST loc stat    .lfile          .lsize          .LDRL(kout)
honeywell     055 30          055611055617    060004004000    000563000002
   program stack                  .lflag          .lflg2          .lpqf
055613000400 stack tally      000140220010    004000500002    000000660010
002756007030 card/callp       rdblk-relinq    logon           extra-buffer
003330007023<list/exec        pgm-in-core     ccall-entry     data-xmit
   null                       pgm-alc-que     vip-term        buffers-full
   null                                       lastio-out      buff-in-core
   null
```

The processor queue display verb, PRQ, can be used to display the linked list of USTs eligible for processor allocation. The display first shows the start of the processor queue emanating from the TSS communications region followed by the entries in the linked list. Each entry in the queue is represented by the user-ID, UST location, and the contents of .LPQF (link in queue).

```
FN??prq
 *processor queue*
first UST address in proc list= 055264
     his         loc=055140         .lpqf= 00131
```

The UOF verb can be used to set a working level of offset for snapshot purposes based upon a user-ID the user has supplied. If the user does not enter the user-ID on the verb input line, the subsystem will prompt the user for a user-ID. The verb provides no acknowledgement if the entry was successful. The contents of the working snap offsets may be verified using the DOF verb (see below).

```
FN??uof honeywell
     or
FN??uof
*enter user-ID* software
```

The DUO verb can be thought of as complementary to the above UOF verb in the fact that both are related to addresses in the dump referenced by a user-ID. The DUO verb can be used to display the UST address for any given user-ID. The terminal user may enter the user-ID following the verb or the subsystem will obtain the verb with a succeeding prompt. No provision is made in the DUO verb processor to handle duplicate user-IDs. The first user-ID found in the linked chain of UST's that matches the desired id will be the entry that is displayed.

```
FN??duo education
*offset for user-ID=education     055430
     or
FN??duo
*enter user-ID* marketing
*offset for user-ID=marketing     056720
```

The COQ verb is similar to the PRQ verb discussed above except for the fact that the memory queue linkage is traversed. The display first shows the origin of the memory queue based upon the start of the linked list in the TSS communication region. Following the start of the list, the entries are displayed showing the user-ID, the UST location, and the contents of the linkage word (.LFLAG).

```
FN??coq
 *memory queue*
 first UST address in mem list = 054665
master          loc=054650          .lflag=056735
sft             loc=056720          .lflag=055445
honeywell       loc=055430          .lflag=000140
```

The TSS allocator keeps a trace of events in the communication region, and the event table may be displayed using the ATR verb. The display consists of the entry pointer as found from the tally word that is used to store the entries and the formatted entries themselves. Each entry is broken out to show the calling location from which the call was made to the allocator, the UST address representing the user for whom service is being performed, the argument indicating which event (type of service) is being processed, a text description of the event, and the user-ID that corresponds to the UST location in the entry.

```
FN??atr
*allocator trace*            last entry index= 02
     callingloc       USTloc     arg          description          user-ID
       023610         055140      3     turn off fileio rdbk     his
       022300         055140      7     rel memory from SS       his
       014667         055140      0     new entry memalc q       his
       025032         055140      2     turn on fileio rdbk      his
       025636         055140      3     turn off fileio rdbk     his
       022300         055140      7     rel memory from SS       his
       014667         055140      0     new entry memalc q       his
       025032         055140      2     turn on fileio rdbk      his
       025636         055140      3     turn off fileio rdbk     his
       023424         055140      2     turn on fileio rdbk      his
       023610         055140      3     turn off fileio rdbk     his
       023424         055140      2     turn on fileio rdbk      his
       023610         055140      3     turn off fileio rdbk     his
       023424         055140      2     turn on fileio rdbk      his
       023610         055140      3     turn off fileio rdbk     his
       023424         055140      2     turn on fileio rdbk      his
```

The TSS executive keeps information about the swapping required to support many users in various cells in the communication region. The related cells may be displayed using the SWF verb. The verb displays many items related to the size and activity of the TSS swap files. The display indicates how many swap files are active, their sizes and growth factors. Following the above general information, each swap file is displayed showing the number of swap ins and swap outs, the file sizes, number of entries for each file, the map showing number of available llinks and starting llink number, followed by the total number of llinks available for each file.

```
FN??swf
*swap files*
#swp files active=04   min swp file size=000220   min grow factor=000044
                          #s          #t           #u           #v
#swapin in progress       0000        0000         0000         0000
#swapout in progress      0000        0000         0000         0000
swap file sizes (ll)      2570        2570         2570         2570
#entries                  0001        0001         0001         0001
#avail ll/strt ll#   02567 000001 02567 000001 02567 00001 00007 00005
total # llinks available    002567       002567       002567      00007
```

The executive trace module (TSSB) keeps a rotating table of event traces in the communication region. The trace table may be formatted and displayed using the TST verb. Each entry in the trace table is broken out showing the original two word contents, the text description represented by the entry type, the user-ID for which the entry is being made (located using the UST address), and the station identifier for those trace entry types that contain the station id. If the entry is the type made by the derail processor, the derail type code is interpreted and shown next to the text description. The next entry to the trace table is located using the pointer NXT> and is derived from the trace tally word used to store entries.

```
FN??tst
*TSS trace table*
          wd1          wd2            description        user-ID stat
      000434000000  055140651013  alloc Q removal     his     6510
      000071000505  054650202011  derail T.STAT       master  2020
      000066000511  054650202011  derail GWAKE        master  2020
      220010660013  055430636516  collect at KEYOU            6365
      220010660012  055430636516  collect at KEYOU            6365
      122125116015  055140651010  command found       his     6510
      000240170730  000004000006  entry line-srv              0000
      122125116015  055140651010  command found       his     6510
      055321055327  055140651012  alloc Q entry       his     6510
      000004000160  055140651011  derail KIN          his     6510
      000033002374  055140651011  derail PASUST       his     6510
      000006000214  055140651011  derail DEFIL        his     6510
      000012000321  055140651011  derail REW          his     6510
      000020000340  055140651011  derail SNUMB        his     6510
  nxt>200070600001  055140651016  collect at KEYOU            6510
      220010660017  055430636516  collect at KEYOU            6365
      220010660013  056720653016  collect at KEYOU            6530
      220010660016  055430636516  collect at KEYOU            6365
      220010660015  055430636516  collect at KEYOU            6365
      220010660014  055430636516  collect at KEYOU            6365
      000232347306  000004000006  enter line-srv              0000
      000233557200  000004000006  enter line-srv              0000
      200070600000  055140651016  collect at KEYOU            6510
      220010660012  056720653016  collect at KEYOU            6530
      000235153616  000004000006  enter line-srv              0000
      000005000015  055140651011  derail RETURN       his     6510
      000430000000  055140651013  alloc Q removal     his     6510
      777767777777  055140651003  system level        his     6510
      000236077103  000004000006  enter line-srv              0000
      103101122104  055140651004  startp primitive    his     6510
      055321055327  055140651012  alloc Q entry       his     6510
      000010000161  055140651011  derail SETSWH       his     6510
      000004000162  055140651011  derail KIN          his     6510
      000006000165  055140651011  derail DEFIL        his     6510
```

wdl	wd2	description	user-ID	stat
000036000747	055140651011	derail FILACT	his	6510
000006001422	055140651011	derail DEFIL	his	6510
000001001543	055140651011	derail DIO	his	6510
000001001543	055140651011	derail DIO	his	6510
000001001543	055140651011	derail DIO	his	6510
220010660011	056720653016	collect at KEYOU		6530
000001001543	055140651011	derail DIO	his	6510
000237303713	000004000006	enter line-srv		0000
000100000171	054650202011	derail T.EXEC	master	2020
000053001153	055140651011	derail SWITCH	his	6510
000014001156	055140651011	derail RETFIL	his	6510
000012001166	055140651011	derail REW	his	6510
000071000454	054650202011	derail T.STAT	master	2020
000005001204	055140651011	derail RETURN	his	6510
000000000000	000000000000	filler entry		

All nonzero I/O queues found in the SSA region of the executive are available for display through the use of the IOQ verb. The display lists the relative negative address (offset from the origin of TSS) of the I/O queue and the status of the queue entry prior to the eleven word entry itself. The format of the I/O queue entry may be found in the system tables manual.

```
FN??ioq
**I/O queues**
766000  bldg 000000000001 000000000000 000355000001 000000000005 036512000013
055622000055 000000000000 636262202020 000237534503 055544055546 055621046365
766013  bldg 000000000001 000000000000 000355000001 000000000005 036516000012
057112000076 000000000000 636262202020 000237202705 057034057036 057111036530
```

For each user on the system, a table of files is kept as a linked list, starting from the UST and threading through the SSA area. The linked list of Available File Table (AFT) entries is broken out using the AFT verb. The display shows the number and names of all open files in the AFT for a given UST.

```
FN??aft

    *aft for user-IDe=sft           #open files=03
          sy**
       knwpatch
       *src
    *aft for user-ID=honeywell      #open files=03
       sy**
       aprntsrc
       *src
    *aft for user-ID=his            #open files=03
       sy**
       pptsrc
       *src
    *aft for user-ID=master         #open files=01
       sy**
```

The SOF verb may be used to set a working level of offset to be used in snapshots. The user may supply a string of up to six octal digits following the verb to indicate the relative address desired for the snapshot. The subsystem will prompt the user for the address if one is not supplied on the verb line. The address will be checked for proper boundary values (i.e., within the limits of the memory assigned to TSS) and for nonoctal digits. No reply is sent to the terminal if the offset has been set accordingly. The contents of the working levels of offsets may be verified using the DOF verb (see below).

```
FN??sof 123
```

The PGD verb is used to obtain information about any of the valid subsystems contained in the list of program descriptors kept in the communication region. The verb requires the user to supply the four character name of the subsystem for which the display is intended. If the subsystem name is three characters (e.g. new) the user must blank-pad his input. The display of the descriptor information contains the size of the program, its load size, seek address, initial load address, entry point, command language pointer, and length of command list. Additionally, the program descriptor permissions bits are interpreted beneath the entry for the subsystem. The abbreviations used to represent the permissions are as follows:

abbreviation	interpretation
spz	special size request
mas	user of T.exec permitted
cmv	use of t.cmov permitted
spo	special product offering
hiu	place SS on #p
tfs	cannot use oldp#
nsy	cannot use at system level
spc	allowed use of get specific
exe	execute permission on I/O
smc	may look at smc
bas	uses basic command list
com	uses common command list
lou	place SS on #q
pch	may be patched
mst	master ss
prv	privileged ss

```
FN??pgd mast
*program descriptor mast*
pgm-size=006756  load-size=006602   seek-addr=000720   ia-load=000154
entry-pt=000154  coml-ptr=007022    #wds-coml=000000   .
mas smc mst
```

The DRL verb displays the derail map, kept in the derail processor (TSSK), at the terminal. The relative addresses of each derail processor are shown next to the name of the derail.

```
FN??drl
**derail map**
        (null)-026142   dio    -023252   kout   -022353   koutn -022341
        kin    -022726   return-022267   defil -024047   abort -027676
        setswh-023051   rstswh-023045   rew    -023704   filsp -023745
        retfil-024213   relmem-027563   addmem-027640   corfil-026146
        snumb -026222   time   -026232   pasaft-025762   termtp-026274
        pdio   -023071   restor-030206   spawn -027204   tapein-022566
        callss-026410   user-ID-026315  term   -026362   pasust-026563
        morlnk-027003   newusr-026507   filact-0243-2   setlno-031052
        sysret-027557   stpsys-027667   status-022214   drldsc-022676
        pasdec-025760   jsts   -031115   cgrout-031651   part   -031737
        grow   -032005   abtjob-032155   consol-032274   switch-031523
        drlimt-033270   jout   -032547   kotnow-022353   objtim-033303
        pasflr-027203   stoppt-023065   save   -034546   task   -033332
        pseudo-035506   prgdes-030146   gwake -033311   ids    -026142
        attri -035610   t.stat-035730   t.goto-026472   t.cmov-035746
        t.linl-036750   t.syot-036241   t.conn-036332   t.cfio-036556
        t.exec-037033   t.rssc-037062
```

The system stores the program registers in the Slave Service Area upon faults/interrupts. To display the register storage stack and tally word, the REG verb may be used. The display shows the contents of the stack tally pointer followed by the four groups of eight words used to store the contents of the registers (X0-X7,A,Q,E,T). Each group of eight words is prefaced by the address in memory and the last entry is marked by a string of ">>".

```
FN??reg
*ssa registers*      register tally ptr(.SSA+1)= 211030000110
01011000     000000000000    000000000000    000000000000    000000000000
             000000000000    000000000000    000000000000    000000000000
01011010     000000000000    000000000000    000000000000    000000000000
             000000000000    000000000000    000000000000    000000000000
01011020     001600000000    204740224451    000000212000    000005000000
             132000000000    060245076340    776000000000    000137645000
01011030   >>011664006000    021271004200    066214004200    000000200026
             066214200026    066214200026    000000000000    000000000000
```

Just as the system stores the contents of the registers at the time of a fault/interrupt, it also stores the IC+I (instruction counter and indicator register values) values in a stack in the SSA. The ICI verb can be used to display the contents of the IC+I stack and its tally word pointer.

```
FN??ici
*ic+i stack tally (.SSA)=210002000000*
260241000220    000036001200    067545104200    000000200026    066214200026
066214200026    000000000000    000000000000    000000000000    000000000000
```

The UAD verb shows the station identifier, UST address, and user-ID for each terminal logged on to the system.

```
FN??uad
   user-ID       lineid  USTloc
sft              6530    056720
honeywell        6365    055430
his              6510    055140
master           2020    054650
```

The slave prefix area (SPA) can contain useful information about the program. Included in the breakout of the SPA are the fault vectors, register store areas, and various other sundry cells. The SPA command produces the formatted SPA display at the terminal.

```
FN??spa
**slave prefix**
        zop/cmnd flt                      memory flt
000000  000000000000    045052710000      000000000000     045057710000
        flt tag flt                       div chk flt
000004  000000000000    045062710000      000000000000     045067710000
        overflow flt                      lockup flt       loc abrt/code
000010  000000000000    045072210000      0000000000000    00000000000
        derail inst                       ckpt addr        fcb
000014  000000000000    045075710000      000000000000     000000000000
        gfrc switch     gelbar timer      gelbar ic+i      gelbar fv
000020  000000000000    00000000000       00000000000      000000000000
        pgm entry       lut/dst code      hist reg ptr     wrapup addr
000024  000000000000    000000000000      000000000000     001127200000
        gelbar info     gecall seq
000030  000000000000    000000000000      000000000000     000000000000
                                          snumb-acty       loader addr
000034  000000000000    000000000000      000000000000     000000000000
        registers-1
000040  001600000000    224740224451      000000000000     000005000000
        registers-1
000044  132000000000    060225076340      776000000000     000132437000
        registers-2
000050  0016000000      224740224451      000000000000     000005000000
        registers-2
000054  132000000000    050245076340      776000000000     000132437000
        eof buffer
000060  001064020100    00000000000       000000000000     00000000000
                                          ident
000064  000000000000    000000000000      000000000000     000000000000
        ident
000070  000000000000    000000000000      000000000000     000000000000
        ident
000074  000000000000    000000000000      000000000000     000000000000
```

The DOF verb allows the terminal user to display the current working offset and any offsets that may have been set using the LVL verb (see below). The display shows all addresses as six digit octal numbers relative to the memory limits of TSS.

```
FN??dof
working offset=055410
lvl-1=000000   lvl-2=000000   lvl-3=000000
lvl-4=000000   lvl-5=000000
```

Within the communication region the allocator keeps several cells that relate to the allocation process. These cells are displayed using the ALC verb. The display includes cells .TALPS, .TATMN, .TAPMU, .TAPMR, .TALPP, .TASWF, A.SD3C, .TSIRC, .TSRRC, A.MBA2, A.SDP8, and .TEBMR.

```
FN??alc
**allocation factors**
        .talps          .tatmn          .tapmu          .tapmr
    026000000000    000000000000    000000000012    000000000113
        .talpp          .taswf          a.sd3c          .tsirc
    000000000004    000000000030    000000000000    000000000000
        .tsrrc          .a.mba2         a.sdp8          .tebmr
    000000000001    000000000000    000000000000    000000000000
```

The LAL verb allows the user to obtain the absolute base address and upper address for the TSS memory region.

```
FN??lal
*lal= 01012000    *ual= 01144000
```

The SSA verb allows the user to display the absolute base address at which the SSAs of TSS begins and the number of SSAs allocated to TSS.

```
FN??ssa
*ssa base addr= 01000000   *#ssa's= 000005
```

Within the derail processor (TSSK), two cells are kept which contain a pointer to the last register storage and the last IC+I values. From the range of addresses shown in the two cells, the current subsystem is related back to a user-ID using the SS base address. From the SS information the user-ID is found and displayed following the register and IC+I values.

```
FN??dic
*last drl reg storage ptr=074040   last drl ic+i=074022   user-ID--his
```

Within the executive a skeleton message is built to inform the console operator of the changing status of the executive. The message may be displayed at the terminal using the STS verb. The status message contains information about the size of the executive, urgent user information, and utilization data computed from data kept in the communication region.

```
FN??sts
*TSS memory sizes 045k=cur.   000k=chg.   070k=max.   021k=swap   006k=lgst.
*TSS urgent users 000 =urg   0000=sta id waiting for  000 sec.    000k=size
*TSS usages  00.03=proc. time  0000  core needed by 000  users   10%=%used
```

With the addition of the deferred user concept, several cells were added to the communication region. These cells may be displayed using the DEF verb. The cells displayed include .TSDMX, .TSDPT, .TSDDT, .TSDID, .TSDSD, .TSDST, .TSDJB, and .TSDGT.

```
FN??def
*TSS deferred cells*
          .tsdmx            .tsdpt             .tsddt            .tsdid
       000000000064      000000000000       000000000000      000000000000
          .tsdsd            .tsdst             .tsdjb            .tsdgt
       000000            000000000000             000000      000000000000
```

The ERR verb displays the group of cells logically related to TSS error processing. Included in the display are the register values stored upon entry to TSSF, the communication region cells pertaining to error counts and values, and the TSSF trace of error entries. Entries in the TSSF error trace are broken out to show the original two words, the UST address, the error code and the user-ID.

```
FN??err
*TSS error cells*
.tereg (registers 0-7,a,g)
000020  064040  055140  074340  766041  212000  000005  000000
026222026232     060205076300
.teric   .terrc     .tlsgb           .tllgb
000000    000000   000000000000     000000000000
   TSSF error trace
      wd1             wd2           USTloc   ccerr   errcod     user-ID
055140000000  060205076300  055140   000000              his
000000000000  000000000000  000000   000000              000000000000
000000000000  000000000000  000000   000000              000000000000
000000000000  000000000000  000000   000000              000000000000
000000000000  000000000000  000000   000000              000000000000
```

The operator has the option to broadcast general messages to all users on the system. To display the last message (if any) sent to the users, a user may use the APB verb.

```
FN??apb
**07.220**test of general apb
```

The executive makes periodic passes through the executive control module (TSSM) and one function that is performed is the servicing of the input queue. The last input queue entry processed by the executive may be displayed by using the TSQ verb. The display formats the three-word group and attempt to interpret the entry beneath the three word portion. The interpretations possible are:

```
interpretations
return batch job
accept new users
accept no new users
status request
change memory size
issue warn msg
get console msg
send console msg
return jout snumb
jdac line return
new task max
date rollover
TSS specific msg
master allowed
master locked out
```

```
FN??tsq
**last TSS input queue entry**
000000000000    000000000000    040000000000
status request
```

At times it may be useful to display the raw images for a UST. The capability to display a UST is provided with the UST verb. The verb processor expects a parameter to be supplied with the verb or following a prompt from the subsystem. The parameter may be either a user-ID or a four-digit line number (to handle duplicate user-IDs). No headings are provided for the UST cells, therefore, the user should reference the TSS executive maintenance document or the system tables manual for exact correlations. The display does not include buffer areas for the UST.

```
FN??ust master
 442162632551  202020202022  000000000360  000050572313  000000000000
 000000000000  000000000000  000000000000  055040042020  000000000000
 000000000000  000000000000  000135000360  056735100030  000000000000
 055031055037  000000000000  000100000000  106002002000  600001002000
 000000000000  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  000000000000  000000000000  000000000000
 054713054721  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  014213000050  000000017411  100000000000
 000000000000  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  000000000000  000000000000  106511000065
 000000000000  000000000000  000000002015  000000413614  000257620000
 000000762273  000000000000  000332012437  000000000000  000000000000
 000000000012  000000000000  000000000006  000000000014  000000000000
 000000000101  000000000000  000000000000  000000000000  000000000000
 000000000000  000000000000  000000000000  000000000000  000000000000
```

The INF verb is provided to the terminal user to serve as a quick reminder when the function of a cell is unknown. To obtain an explanation of what a cell's function is, the user may enter the verb followed by the logical name (from the macros) of the cell. The display can provide explanations for logical names in the SSA, UST, and TSSA communication region.

```
FN??inf .tcfil
.tcfil  -file codes for program load files
```

The working levels of offset may be zero-cleared using the CLR verb. No reply is sent to the terminal upon completion of the task.

```
FN??clr
```

The SMO and DMO verbs both function using the TSS module identifiers as parameters (e.g. A, B, C...M, N, O). The SMO verb allows the user to set a working offset which points to the start of the module as specified by the user. The DMO verb allows the user to obtain the offset for a given module based upon the input.

```
FN??smo f
FN??dof
working offset=011504
lvl-1=000000  lvl-2000000  lvl-3=000000
lvl-4=000000  lvl-5=000000
FN??dmo f
*offset for module f   011504
```

The snap verb ,SNP, provides a snapshot capability to the terminal user. The user may snap locations in the dump using different levels of offset (either working or offset levels). The options required by the verb processor take the following form--SNP FLD1, FLD2, FLD3. The first field is the starting address of the snap, the second field is the number of words to snap, and the third field is the optional level of offset to be applied to the snap. The LVL verb assumes a parameter in the range 1 to 5 will follow the verb. The LVL verb is used to move the current working level of offset to one of five levels of offset.

```
FN??snp 0,10  (assuming level set by SMO verb above)
*snap*
000000  070600020002  070600050201  001430753200  011515710200
000004  001440554200  011512710200  001430753200  040107221203
FN??lvl 4
FN??dof
working offset=011504
lvl-1=000000  lvl-2=000000  lvl-3=000000
lvl-4=011504  lvl-5=000000
FN??snp 0,10,4
*snap* l=4 o=011504*
000000  070600020002  070600050201  001430753200  011515710200
000004  001440554200  011512710200  001430753200  040107221203
FN??sof 123
FN??snp 0,1
*snap*
000000  000004056720
FN??snp 0,1,4
*snap* l=4 o=011504*
000000  070600022001
FN??dof
working offset=000123
lvl-1=000000  lvl-2=000000  lvl-3=000000
lvl-4=011504  lvl-5=000000
```

The history registers captured as part of the header record may be displayed using the HIS verb. If the dump occurred as a result of a hardware failure that was trapped in the history registers, the formatted contents of the registers are displayed. Included in the display are CPU#, fault type, faulting instruction, register contents, and the 16 cycles of the OU/CU. During the tests of the software, no faults within TSS were attributed to the type of error that could be captured in the history registers, therefore no example exists.

The gates in the system are gathered as part of the header record information. The status of the gates may be displayed using the GAT verb. The display shows all closed gates in the system.

```
FN??gat
**closed gates**
     .crlgq
```

Based upon the .CRLAL and .CRSNB tables captured in the header record information, the MEM verb displays the snumb, absolute lower address, and program number of those jobs in execution at the time of the failure.

```
FN??mem
*memory map*
snumb       lal        pgm#
$CALC     000136000     01
$PALC     001174000     02
$SYOT     001316000     03
$RTIN     001242000     04
TSS       001012000     05
$FSYS     000144000     12
```

Within the executive several cells pertain to the "current" user. The cells include .TCLOC, .TESSB, .TELAL and .TCLIN. These cells are displayed along with the user-ID through the use of the CUR verb. The user-ID is located using the base of the current user.

```
FN??cur
   .tcloc          .tessb          .telal          .tclim          user-ID
074000000000    074014055140    074000000011    014000000000      his
```

The executive manages a linked list of cells that indicate the availability of work space for UST usage. The linked list may be displayed using the CWK verb. The display locates the hole and indicates the size of the area.

```
FN??cwk
**ust mgmt core-hole list**
     loc-hole    size
     057210      000570
```

The executive now has the ability to utilize more than one processor via the sub-dispatch queue. The information relating to the use of the sub-dispatch queue is contained in two areas--1)the communication region (queue header) and 2) an area beyond TSSO and prior to the USTs (the entries). The display of the SDQ verb formats the queue header information and follows that with the analysis of each of the three threaded lists--1) the ready chain,2) the fault chain, and 3) the available chain.

```
FN??sdq
**sub-dispatch queue header**
    gate        #pro all/dsp  tod at disp   time quantum  #disp
016463   open   000004000000  000240204633  000000002400  000000002247
proc time SS    q busy count  ready chain   fault chain   avail chain
000003055375    000000000000  000000000000  000000000000  054420000000
    temp        ready count   rdy limit
000000000000    000000000000  000012400004
**ready chain empty**
**fault chain empty**
*avail chain*
    user-ID      USTloc  ss-bar   fault/inter
his              055140  074014   inter
his              055140  074012   inter
```

The executive maintains a list of queued events called the executive service queue. The linked list is displayed using the ESQ verb. The user-ID and UST address are displayed for any entries found.

```
FN??esq
**executive service queue empty**
```

The executive maintains an area in the communications region to hold the message directed to a specific user-ID or station. The TSM verb can be used to display the specific message cells. The display shows whether the message was destined for a user-ID or station id and follow that indication with the message itself.

```
FN??tsm
*TSS specific msg for sft
**07.222**test of specific message for software
```

The EMM verb allows the terminal user to see the formatted display of the MME .EMM trace table kept in TSSM. The trace tally pointer, is displayed first followed by the entries themselves. Each entry is interpreted to show the original contents, user-ID, relative address of the call to the trace routine, and the module from which the call was made. If the value contained in index register 2 can be identified as a UST pointer, the user-ID is interpreted as part of the display. The trace tally pointer value is used to mark the next entry with a NXT>.

```
FN??emm
**emm trace**    tally ptr-046252000100
           entry        user-ID(x2->)      loc of call
        012150224740                      000444 TSSF
        012150224740                      000444 TSSF
        012150224740                      000444 TSSF
        012150224740                      000444 TSSF
   nxt> 012150224740                      000444 TSSF
```

To exit from the subsystem the END verb or the PRT verb may be entered by the user. The END verb displays the ratio of disk I/O hits/misses prior to issuing the DRL RETURN to complete processing. The PRT verb asks for an $ IDENT image (conforming to site standards) to be used as a header to the printer report produced. The reply following the request for an ident image will be the snumb assigned to the backdoor sysout job producing the report. The printed report contains some assorted header information followed by the dump body. Throughout the dump several pointers are inserted to identify the origin of the TSS modules and the UST's.

Purpose

The UCAS command transliterates all lowercase ASCII characters to uppercase ASCII.

Format

UCAS

Discussion

This command applies to keyboard/display type devices only. See the LCAS command.

Purpose

The WRITE command directs output to a designated tape cassette.

Format

WRIT[E] TAPE [n]

[n] ::= 1 or 2, default value is 1.

Discussion

This command can only be used on 7700 Series VIP (device 13, 14 or 15 octal).

SECTION VII

LOADING USER SUBSYSTEM PROGRAMS

User subsystem programs may be executed in the Time Sharing System using one or more of the loader functions described in this section. LODT and LODS support two of the loader functions and enable execution of a user program or resident TSS subsystem in a debugging environment. The Command Loader is a default subsystem that is invoked whenever an unrecognized command is given, either at system selection level or in line-numbered build mode. The Command Loader interprets this "invalid command" as an H* cat/file descriptor and will attempt to access, load, and pass control to the associated program.

COMMAND LOADER SUBSYSTEM

The Command Loader Subsystem is invoked by the TSS executive whenever an unrecognized command is given, either at system selection level or in line-numbered build mode. This subsystem is nearly identical with LODX, except that (1) no questions are asked of the user, and (2) the input is construed to be the file description of an H* file that is to be loaded and executed. The descriptor must conform to one of the following conventions:

1. catalog/filename is taken as is. If the cat/file descriptor is qualified by a user-ID, the reference is to the named file in the specified user's catalog. Passwords, permissions, and up to three levels of subcatalogs may optionally accompany the descriptor. Default permissions will be read only.

2. /filename implies user-ID/filename. The named file emanates from the user's own catalog. As with the above, passwords, permissions, and subcatalogs may be specified.

3. filename, delimited by a blank or carriage return, implies CMDLIB/filename. CMDLIB is a special system master catalog (SMC) entry, analogous to the existing LIBRARY SMC, which may contain installation-coded subsystems. Each such subsystem must be a quick-access H* file having general or specific read or execute permission, with a name consisting of the first four (or fewer) characters of its associated command. Delimiters other than blank or carriage return should not immediately follow the filename specification, as a loading failure may result. Although the file may be passworded, the password cannot accompany "filename." Note that the subsystems resident in CMDLIB may be coded in FORTRAN, ALGOL or JOVIAL, as well as GMAP. Therefore, this convention permits an installation to easily install its own subsystems.

A blank following an alphanumeric character string terminates the input scan. Blanks are ignored if immediately preceded by a delimiter.

The character string (conventions 1 or 2 above) immediately following the last slash (/) encountered in a file description is placed in the UST I/O buffer via DRL PSEUDO. For example:

*JOEDOE/JSTS 1234T

A DRL KIN, if executed by the loaded program, would receive the character string,

JSTS 1234T

Use of the colon (:) overrides this feature. See "Command Loader Usage" below.

In the following example, a new subsystem to create temporary files is desired. The subsystem is to be invoked by a new command, "CREATE", which may optionally be accompanied with a filename, size and mode indicator. If not specified, it is assumed that the "CREATE" subsystem will request these parameters.

*CREATE TMPFIL1,3,RANDOM

The TSS executive, not recognizing the command "CREA", invokes the Command Loader subsystem. The Command Loader input line scan, upon encountering a blank with no preceding slash, makes a file access with the following description:

CMDLIB/CREA,R

If the access is successful, the program is immediately loaded and control is transferred to it. Thus, a new command has been introduced by simply placing the subsystem H* file in the CMDLIB catalog under a name corresponding to the first four characters of the command itself.

The Command Loader Subsystem can be utilized to initiate a command file application.

COMMAND LOADER USAGE

Conventions, "catalog/filename" or "/filename", optionally permit the user to specify a particular element of a multielement H* file to be loaded. This specification, if present, must immediately follow the file descriptor with an intervening semicolon(;). If multiple elements exist on the H* file and this field is not specified, the Command Loader assumes that the H* file contains an overlay structure and searches the catalog block(s) for the main link, identified by the name "//////". In all cases, the Command Loader always loads the file control block element of an H* file, if one exists.

A third field (or the second field, if an element name is not specified) may accompany the input and must be preceded by a colon(:). If present, all characters of the input line up to and including the colon itself are effectively deleted from the key I/O buffer resident in the user's UST. Thus, a subsequent DRL KIN, when issued by the loaded program, will only receive that portion of the original line that immediately followed the colon.

The Command Loader issues the message, "009-SYSTEM UNKNOWN" or "COMMAND UNKNOWN" (whichever is appropriate) when any loading failure occurs. The user may determine the reason for the failure by requesting LODX to load the same file.

The H* file to be loaded is always accessed with an alternate name of the form ".HS.", where "S" assumes a value in the range 0-3, inclusive. The choice of this value is a function of the current CALLSS pushdown level. Prior to accessing the specified file, both the file itself and the alternate name (.HS. at the same level) are deaccessed. Unless the H* file contains an overlay structure, .HS. is deaccessed upon completion of loading.

In order to force the system to invoke the Command Loader when the first four characters of a cat/file description duplicate a known TSS command, the user can prefix the cat/file description with any printable character(s) that are not syntactically legal for a description, such as blank, !, ?, etc. The Command Loader ignores all leading characters of the input line until it encounters a letter, digit, dash, period, or slash. If it is necessary to ignore leading characters, the Command Loader effectively deletes these characters from the UST I/O buffer via DRL PSEUDO. This is for the benefit of subsystems that execute a DRL KIN to retrieve the last line of input.

The Command Loader facility is also available through DRL CALLSS by specifying the argument name "CMDL" (or any unique name). A program that invokes the Command Loader in this manner must ensure that the I/O buffer in the UST is properly prepared for examination by the loader. The DRL PSEUDO service function provides a convenient means for accomplishing this.

LODX

As with the Command Loader function, a cat/file descriptor accompanied by an optional element name and/or the partial line-delete (colon) field is required. If this information is not specified on the same line as the LODX command itself, a request is issued for it. Upon completion of loading, the user is given the opportunity to:

1. Apply octal patches, either from the keyboard or a file.
2. Save the loaded program, either back on the original H* file or on a specified file.
3. Place the loaded program into execution.

POST-LOADING OPTIONS

The following message is issued to the user upon completion of the loading function performed by LODX, LODT or LODS.

PATCH, SAVE OR RUN?

Only the first character (P,S, or R) of the response is necessary. If a null response is given (carriage return only), the loading function is terminated and user returned to the system selection level or build mode.

PATCH

LODX responds with a "?" indicating readiness to accept the first patch. The patch data must consist of a 1- to 6-digit octal address, delimited by a blank, which in turn must be followed by any number of 1- to 12-digit octal fields (the patch data), separated by commas. Successive question marks are issued to obtain patches until receipt of only a carriage return, "*", or "D". A carriage return causes reissuance of the "PATCH, SAVE OR RUN?" query, while an "*" or "D" causes control to be passed to the loaded program.

PATCH Filedescr

The specified file is used as the patch source. The format of the file is exactly the same as a series of patches entered from the keyboard. A patch file created by the text editor may also contain the "*" or "D" indicator to enable program execution. If an end-of-file or any error is encountered, the "PATCH, SAVE OR RUN?" query is reissued.

The PATCH function of LODX, LODT, and LODS accepts patches that are formatted for the $PATCH section of startup. A blank terminates the patch data and allows comments and/or module catalog names to be included on the line containing the patch(es); e.g.

1	8	16	32	73
243	OCTAL	5600004	TZE 5,IC	.TSACC

The usefulness of this feature is apparent in that a patch file can be constructed, tested and subsequently JPUNCHed for inclusion in the startup deck.

SAVE

The loaded program is stored back on the H* file from which it was obtained. Note that the file now contains a single program element, regardless of how many elements were initially present.

SAVE Filedescr

If the specified file exists, LODX saves the loaded program in H* format on this file. If insufficient space exists, an attempt is made to grow the file or, if the file does not exist, it is created for the user at this time. The trace package is not included on the saved file when LODT or LODS has been specified.

SAVE Filedescr;Progname

The loaded program is appended as an additional element on the specified file with a name corresponding to "progname". The name must consist of 1-6 alphabetic and/or numeric characters (period or dash is also permitted).

RUN

The loaded program is entered for execution at the entry address specified in the control block of the H* file.

RUN nnnnnn

Same as above, except an alternate octal entry address, nnnnnn, is desired by the user.

LODT

The LODT subsystem provides a debugging environment for a user program resident on an H* file. As with LODX, the H* file is loaded and the user given the opportunity to PATCH, SAVE OR RUN. In addition, however, a copy of the trace package is appended to the resulting load, and TRACE is provided with the program's true entry address in its linkage register (X1). When the RUN command is given, LODT transfers control to the trace package. TRACE is thus initially given control, and when its first "R" command is exercised, program execution begins. If the trace mechanism is engaged before issuing the "R" command, the user's program will be executed in a controlled environment.

LODS

LODS is similiar to LODT, except that a specified TSS subsystem is loaded and bound with the trace package instead of an H* file. This capability is primarily intended for those responsible for subsystem maintenance and site system personnel. The command associated with the desired subsystem, followed by any of its necessary parameters may accompany the LODS command. If not specified on the same line as the LODS command, this information is requested from the user. As with LODX and LODT, an opportunity is given to PATCH, SAVE (filedescr required) OR RUN. Prior to relinquishing control, LODS removes all characters of the input line that prefix the command word (via DRL PSEUDO). This would normally be the LODS command itself and its terminating delimiter. Thus, for example, the following use of LODS would result in loading the LIST subsystem for debugging purposes:

```
LODS LIST FILEX(100,200);FILEY
```

NOTE: The LODT and LODS commands permit the load origin of the Trace Package to be specified. (See the Debug and Trace manual.) This specification must be preceded by a semicolon and requires the format, TRACE-nnnnnn, where nnnnnn is the desired octal address at which to load the Trace Package. Thus, to load a program with the Trace Package origin at location 14000:

```
*LODT JOE/JSTS;TRACE-14000:JSTS 1234T
```

This feature is useful for debugging overlay structures or for programs that utilize core beyond that defined by the program size at load time. If a program element must also be specified, it may either precede or follow the origin specification.

GENERAL RULES REGARDING ALL LOADER FUNCTIONS

Upon completion of loading, location 31 (decimal) of the program's slave prefix is initialized to reflect the unused space (hole) between the last location loaded (bits 0-17) and the end of allocated core (bits 18-35). All loader functions that load from an overlay-structured H* file always leave the file in the AFT. The ASCII name (or alternate name) of this file can be found in the slave prefix, words 10 and 11 (decimal). The remainder of the prefix area, in addition to any core allocated to the program that was not initialized during the loading process, is cleared.

The default permissions used by all loader functions for accessing H* or patch files are determined as follows:

1. READ and WRITE permissions are requested if the file description either (1) is not qualified by a user-id, or (2) is specified with the SAVE option.
2. READ permission only is requested in all other cases.

An explicit request for execute-only permission will be honored by both the Command Loader and LODX functions. However, LODX does not issue the request for options upon completion of loading. Prior to passing control to the program, all loader functions execute a DRL OBJTIM, notifying the TSS Executive that READ requests on files accessed with execute-only permission can no longer be honored.

LODX and LODT deaccess the H* load file (if necessary) prior to attempting to access it, unless the file description consists solely of a 1- to 8- character filename. The file is not deaccessed upon completion of loading unless execute permission was specified (LODX). Note that none of the loader functions permit an alternate name specification to be given for a cat/file description.

"LODX" permission must be granted to users, either selectively or collectively, by the master user for use of the Command Loader, LODX and LODT functions; however, permission is not required for use of the Command Loader when the "filename" (CMDLIB reference) convention applies. Similarly, "LODS" permission must be granted to those requiring use of the LODS facility. LODS permission implies both LODX and LODS permission. Note that since LODS is a privileged subsystem, authorization to utilize it should be judiciously granted.

SUBSYSTEM DUMP FACILITY

Dump Procedure

If the user wishes his subsystem to be dumped to a permanent file when an exception condition occurs (or when, at his discretion, he calls for an abort via DRL ABORT) he does the following:

1. Creates a linked file named ABRT of sufficient length to hold his entire subsystem.

2. Before calling the subsystem into execution, accesses the file named ABRT. This can be done with the ACCESS subsystem, GET command, etc.

The subsystem will now be dumped to this file when either a fault occurs that the subsystem does not handle or a DRL ABORT is executed by the subsystem. After this occurs, the user can inspect his dump with the subsystem called SABT (Scan Abort File), described below.

If the user does not have an ABRT file in his AFT at the time the exception condition occurs, the TSS Executive will create a temporary file of sufficient size to contain the dump. The ABRT file is released at logoff time without regard to disposition; i.e., the user is not given the opportunity to make it permanent as for other temporary files.

SABT (Scan Abort File) Subsystem

When a fault occurs in a subsystem that does not handle such faults, or a DRL ABORT is executed, the aborted subsystem is copied to the ABRT file. By means of the SABT subsystem, the user can scan the ABRT file by snapping portions of it at the terminal.

SABT is called as a system selection or while in line-numbered build mode:

```
*SABT
OFFSET?
```

The user may specify an offset to be added to all addresses requested. Designation of areas to be snapped can be given as in the following examples (all numbers are octal and will have offset, if any, automatically added to them).

	Meaning
?1235	snap 1 word at 1235
?172,14	snap 14 words starting at 172
?2354-2367	snap from 2354 through 2367
?(carriage return)	done, return to calling level.

Output is typed in the following form:

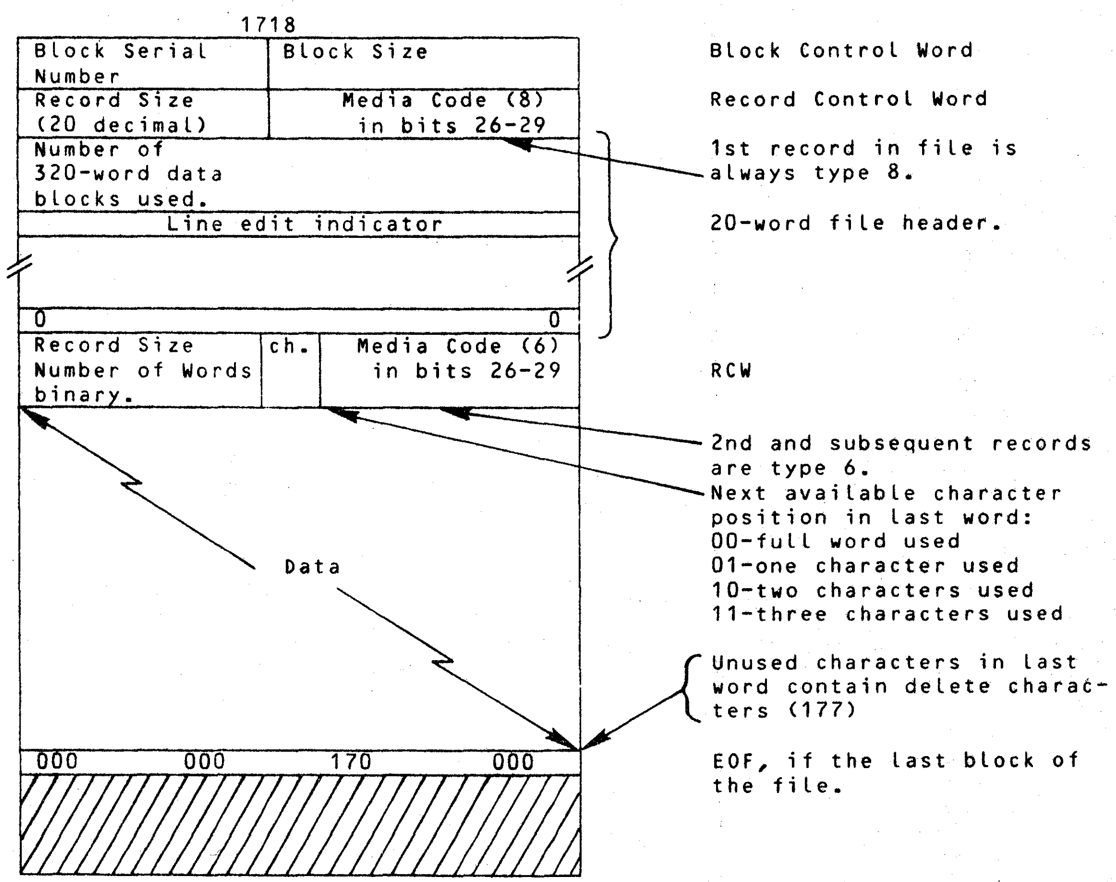
loc word1 word2 word3 word4

It is possible to have SABT request that the dump be printed at the central site by responding "W" to the OFFSET? question. Upon receipt of this response, SABT requests the IDENT image and constructs a batch CONVER activity which is initiated via DRL SPAWN. By responding "w xx" (where xx is a two character remote station identifier, e.g., AB) to the OFFSET? question, the batch CONVER activity will route its output to the remote destination.

SOURCE (SRC) FILE FORMAT

The standardization of source-text files allows more than one system to process these files. For example, using a standard file format allows EDITOR to operate on BASIC text. All text files are maintained in ASCII format. They are linked files that contain block and logical record control words that allow the files to be accessed by File and Record Control. The standard source file used by Honeywell-released subsystems (the current file) is named *SRC. Its format is as follows:

Initial 320-word block:



Block Control Word (BCW) is the first word of each 320-word data block and contains two binary values as follows:

bits 00-17	Binary equivalent of block serial number -- the sequential number of this physical record, beginning with 1.
bits 18-35	Binary equivalent of block size -- the size of the block in words, not including the BCW.

Record Control Word (RCW) - Records within each block are variable in length, and each record begins with a record control word. The contents of the RCW for nonpartitioned files are:

bits 00-17	Binary equivalent of record size in words, not including the RCW. If the file is assigned to disk and this value is zero, bits 18-23 are interpreted as a file mark analogous to a tape end-of-file marker.
bits 18-19	Next available character position in last word. The field is interpreted as: 00 = full word (four characters) used 01 = one character used 10 = two characters used 11 = three characters used In all cases, the two bits indicate the character space and may be used in the formation of a tally word. Any unused character positions will contain a delete character (octal 177).
bits 20-23	Not used unless bits 0-17 are zero, in which case bits 18-23 contain the specific file-mark characters. The standard EOF character is octal 17.
bits 24-25	Zeros.

bits 26-29 Record media code--

0 - Print-line image with no slew (BCD)
1 - Binary record (e.g., FORTRAN binary record, COMDK etc.).
2 - Hollerith card image (BCD)
3 - Print-line image (BCD)
4 - Reserved for user.
5 - TSS ASCII file format (old format)
6 - ASCII Standard System Format
7 - ASCII print-line image, with slew control word
8 - TSS information record
9-15 - Undefined

Media Code 6 is used in TSS except the first record which is Media Code 8. Normally the other codes are not used by TSS.

bits 30-35 Report code.

Second and succeeding 320-word blocks:

0	1718	
Block Serial Number	Block Size	
Record Size Number of words (binary)	char.	Media Code 6
Data		
Record Size Number of words	char.	Media Code 6
Data		
000 000	170 000	

EOF, if last block of file.

The ASCII text consists of strings of 9-bit characters. A character string does not extend from one block to another. A Record Control Word containing 000000170000 (octal) is used to indicate EOF.

SY* FILE FORMAT

The SYT* file format is the same as the SY** format.

All nonempty records except the last:

	0	1718
Rec. Cont. Word	Number of Words	Relative Block Count
63 Words	Record Control Word (RCW)	
	9-bit ASCII characters	
	RCW	
	9-bit ASCII characters	
	RCW	
	9-bit ASCII characters	
	Unused	

Count begins with zero

Final Nonempty Record:

	0	1718 35
	Number of Words	Relative Block Count
63 Words	RCW	
	9-bit ASCII characters	
	RCW	
	9-bit ASCII characters	
	170000	
	Unused	

Empty record (a command word was the first line in input buffer)

0	17 18	35
Number of Words	Relative Block Count	
	170000 (EOF)	
	Unused	

NOTE: An empty record may or may not be the first record in file. The Record Control Word (RCW) is described earlier under "Source (*SRC) File Format."

TAP* FILE FORMAT

TAP* is the punched paper tape (PPT) collector file that contains the unedited PPT input. It is a random file, with a maximum of two links.

Format from mass storage devices - 64 words/block:

	C	17 18		33	34	35
	Number of Words	Relative Block Count				
31 wd. max.	m			x	y	0
		9-bit ASCII Characters				
31 wd. max.	m			x	y	0
		9-bit ASCII Characters				
		Unused				

m = character count of input data block (<120) - may be zero
x = 1 if timing error occurred
y = 1 if last block

TIME SHARING DEBUG TRACE PACKAGE

The debug/instrumentation capabilities available to the time sharing users include the following.

1. A trace mechanism is capable of collecting and/or displaying information as it steps through the program one instruction at a time.

2. Commands SAVE and RESTORE permit the current state of the program to be saved and subsequently restored. This feature effectively provides a means to back up and start over again.

3. Multiple commands, separated by a slash (/), can be specified on a single input line.

4. Continuation input is possible by ending an input line with a delimiter.

5. Mnemonic operation code and modifier interpretation can optionally be requested with memory snapshots.

6. The PATCH command permits a repeat count for patching a contiguous area of memory with the same datum word.

7. The LOCATE command can be used to list the absolute addresses of all SYMDEFs defining subprogram entry points. Alternatively, only selected SYMDEFs can be listed.

8. The CALL command permits the user to invoke any selected subsystem. When the subsystem terminates, control is returned to the trace package.

9. The break key can be used to either initiate a manual breakpoint or terminate output being produced by a trace package command.

10. Programmed breakpoints (those established by the B command) can be conditional; i.e., such a breakpoint is serviced only if a prescribed condition exists.

11. The FIND command can be used to search memory for a specified data pattern and print the address at which the pattern was found. An optional mask and/or repeat count can accompany the command.

12. The EXECUTE command permits a specified number of target program instructions to be executed in the controlled environment provided by the trace mechanism, followed by a return to command level.

13. The package has a built-in user error detection capability. For example if the user attempts to snap or patch memory outside the bounds of allocated memory, or enters an invalid command parameter an error message is printed.

Binding Trace Package With Target Program

The trace package can be bound with the target program by including an object deck in the General Loader activity which generates the LODX H* file. This technique necessitates some means of invoking the package at execution time. Several alternatives exist,

1. Include a $ ENTRY TRACE control card in the General Loader activity. This causes control to be passed to the trace package by LODX upon completion of loading. The actual entry address must be supplied with the first RUN command, when it is finally issued.

2. Incorporate one or more calls to the trace package in the source program before assembling or compiling. This is accomplished by using a call statement, as follows:

```
CALL     TRACE
```

Programs coded in GMAP can utilize the following sequence for linkage:

```
SYMREF  TRACE
      .
      .
      .
TSX1    TRACE
    (or)
XED     TRACE
```

Regardless of the linkage choice, the next RUN or EXECUTE command causes execution to be resumed at the location following the TSX1 or XED. The trace package can accommodate both types of linkage, using the same entry point for each type. The entry sequence is:

```
      SYMDEF  TRACE
            .
            .
            .
      EVEN
      TSX1    0,IC*
TRACE NOP     **,DU
            .
            .
```

The entry point (TRACE) is forced into an odd location. This is no problem for the TSX1 which transfers to the NOP. The XED, however, having an odd effective address, causes the instruction pair (TRACE-1,TRACE) to be executed. Since the first instruction of the pair is a TSX1 with IC* modification, its effective address is TRACE.

If the linkage register (XR1) is in use and must be preserved, the following sequence should be used to preserve it:

```
STX1   TRACE
TSX1   TRACE
   (or)
STX1   TRACE
XED    TRACE
```

3. Utilize the patch function of LODX to patch in a TSX1 (or XED) to the trace package.

A special loader, LODT, is available to: load an existing H* file, append a copy of the trace package to the loaded program and simulate a call to it from the program's actual entry point. This technique is possible, since the trace package has been implemented in floatable code and can be loaded anywhere in memory without relocation.

Command Language Usage

Since the trace package has been bound to the target program and placed in execution, the following message is issued when the package is invoked at its entry SYMDEF (TRACE):

```
NNNNNN: FUNCTION?
```

This indicates readiness to accept the first command. The address, NNNNNN, is the location of the invoking TSX1 or XED. Unless an offset has been previously established, the indicated address is absolute. After performing a specified command, or if the break key is used to interrupt it before completion, the trace package issues a question mark (?), which is an implicit request to enter the next command. This process continues until a RUN, EXECUTE, ABORT or TERMINATE command is exercised.

The commands can be grouped in two general categories according to type. Type 1 commands require a single letter function identifier although some must be accompanied with parameters. Type 2 commands require a word response. Although the full word can be specified, only the first four characters are required.

In most cases, both types of commands require accompanying parameters, while with some, parameters are optional. Unless otherwise stated, the following rules apply to parameter specification:

1. All numeric parameters (addresses, repeat counts, etc.) must be specified in octal.

2. Parameters must be separated from one another by a single comma, or by one or more blanks. If the first parameter is numeric, it can immediately follow the command with no intervening delimiter.

3. Alphabetics can be specified in either uppercase or lowercase.

4. All nonprinting characters except the carriage return are ignored.

5. If more parameters must be specified than can be contained on a single line, the line can be terminated with a delimiter. This causes a request for continuation input to be issued to the user. The terminating delimiter can be a comma, blank, dash or semicolon, whichever is syntactically correct for the command.

In the command descriptions which follow, the conventions listed below have been adopted.

A1 - the first one- to six-digit octal address or data specification.

An - the nth such one- to six-digit octal quantity.

D1 - the first one- to 12-digit octal data word specification.

Dn - the nth such one- to 12-digit octal data word.

n - one- to six-digit octal repeat count.

If fewer than the prescribed number of digits are specified, leading zeros are assumed and the quantity is right-justified.

Nontrace Commands

ABORT (TERMINATE EXECUTION VIA DRL ABORT)

The ABORT command can be used to terminate execution abnormally, with an immediate return to system selection level. If an abort file has been previously accessed, the Time Sharing Executive dumps the contents of allocated memory to this file. The dump can be scanned with the SABT subsystem.

B OR BA (ESTABLISH BREAKPOINT)

The BA command can be used to establish one or more breakpoints in the user's program. A breakpoint is a location where the trace package regains control when the instruction at that location is executed. At each time, the following message is issued:

NNNNNN: BREAKPOINT

Where: NNNNNN is the address of the breakpoint, minus the offset if one has been previously established.

Following issuance of this message, the trace package responds with a question mark (?), which is an implicit request to enter the first command. When the next RUN or EXECUTE command is given, the original instruction at the breakpoint location is executed.

A breakpoint can be established at any location that contains a legal instruction (DRL and EIS[1] included); however, the location cannot be one that is influenced by a repeat-type instruction.

Permissible forms of the command are:

Form	Meaning
B A1,A2,...An	Break at each effective address offset+Ai.
BA A1,A2,...,An	Break at each absolute address Ai.

In some instances, it is desirable to have a breakpoint serviced only if certain conditions exist. Such a breakpoint can be specified by appending a relational operator of the form .REL. to the address, Ai. The conditions are based on indicator register status at the time the breakpoint location is executed. The following table describes the various operators:

Operator	Causes Breakpoint Servicing Only If
.EQ.	zero indicator is on
.NE.	zero indicator is off
.LT.	negative indicator is on
.LE.	negative or zero indicator is on
.GT.	negative and zero indicators are off
.GE.	negative indicator is off
.LLT.	carry indicator is off
.LLE.	carry indicator is off or zero indicator is on
.LGT.	carry indicator is on and zero indicator is off
.LGE.	carry indicator is on

Examples: B1723,1727.EQ.,1730

BA 423

C (ENABLE CONTROL VIA BREAK KEY)

The C command can be used to re-enable use of the break key for initiating manual breakpoints and/or terminating output being produced by subsequent commands. Normally, the user does not need to exercise this command because break control is automatically enabled by the trace package when (1) the package is first entered, (2) any subsequent use of the break key is made, (3) a breakpoint is executed, (4) any derail instruction is executed by the trace mechanism, and (5) a CALLSS command has been completed. Thus, its use is limited to cases where the target program has altered the break vector in the slave prefix area.

---

[1] Extended Instruction Set (EIS) refers to extensions to the original GCOS instruction set. EIS is included in the standard instruction repertoire of Models 6025, 6040, 6060, and 6080 of the Series 6000 system and for all models of the Series 60 Level 66 systems.

CALLSS (CALL SUBSYSTEM)

The CALLSS command can be used to initiate an internal call to any desired subsystem via a DRL CALLSS. If a carriage return immediately follows the command, the trace package responds with the question,

SYSTEM?

The user should, at this time, respond with the desired subsystem name, followed by any parameters required by the subsystem. Alternatively, this information may be specified on the same line as the CALLSS command. In this case, simulated keyboard input (via DRL PSEUDO) is performed to effectively remove the CALLSS command and its delimiter from the input line. This benefits those subsystems that perform a DRL KIN to obtain the last line of input.

When the called subsystem terminates normally, or if the break key is used to abort it, the trace package issues the following message and resumes accepting commands:

CALL COMPLETED

Multiple commands cannot be specified on the same input line following the CALLSS command.

Examples: CALL

SYSTEM? ACCE CF,/ABRT,B/40,100/,R

CALL JSTS 2507T

D OR DA (DELETE BREAKPOINT)

The D or DA command can be used to delete one or more previously established breakpoints in the user's program. Deleting a breakpoint consists of removing the appropriate entry from the trace package breakpoint table and restoring the original instruction at the breakpoint location.

Permissible forms of the command are:

Form	Meaning
D	Delete the current breakpoint only.
D A1,A2,...,An	Delete the breakpoint at each effective address, offset+Ai.
DA A1,A2,...,An	Delete the breakpoint at each absolute address, Ai.
D ALL	Delete all breakpoints.

A request to delete a specified breakpoint which does not exist is ignored.

DEC (DECIMAL-TO-OCTAL CONVERSION)

The DEC command can be used to convert a decimal number to its octal equivalent. Decimal numbers up to 11 digits can be converted. The only permissible form of the command is:

Form	Meaning
DEC n	Convert the given decimal number to octal and display it.

Example: DEC 9361

E (EXECUTE INSTRUCTIONS)

The E command provides the capability to execute a specified number of target program instructions, starting with the next instruction to be executed. Execution is performed in the controlled environment provided by the trace mechanism. If the trace mechanism is currently engaged, it is temporarily disengaged until the EXECUTE command has been performed. Upon completion of the command, the "NNNNNN: FUNCTION?" message is issued, indicating the new value of the IC (Instruction Counter) and readiness to accept the next command.

Permissible forms of the command are:

Form	Meaning
E	Execute the next instruction only.
E n	Execute the next n instructions.

Multiple commands cannot be specified on the same input line following the EXECUTE command.

F OR FA (FIND DATA PATTERN IN MEMORY)

The F command can be used to find the location(s) of one or more occurrences of a specified data pattern (D1) in allocated memory, with the search commencing at any designated location (A1). An optional mask (D2) can be provided to enable comparisons only on selected bit positions. If provided, bit positions of D2 that contain a 1 cause the corresponding bit positions of D1 to be ignored during the search. If the mask is not specified, comparisons are based on a full 36-bit word.

Permissible forms of the F command are given below. The effective starting address for each form given is offset+A1. In all cases, FA can be substituted for F if the search is to start at absolute location A1.

Form	Meaning
F A1,D1	Find the first occurrence of D1, starting at location A1.
F A1,D1;n	Find the first n occurrences of D1, starting at location A1.
F A1,D1;*	Find all occurrences of D1, starting at location A1.
F A1,D1,D2	Find the first occurrence of D1 masked by D2, starting at location A1.
F A1,D1,D2;n	Find the first n occurrences of D1 masked by D2, starting at location A1.
F A1,D1,D2;*	Find all occurrences of D1 masked by D2, starting at location A1.

If the search is successful, each address at which the data pattern, D1, is found are displayed. The addresses are relative to the offset, provided one has been previously established. In addition, the data content at each address is displayed, provided a mask is specified.

If the search is unsuccessful, the following message is issued:

PATTERN NOT FOUND

Examples: F1310,56060062056 (find ASCII ".02.")

FA110,2000,777777000777;* (find all derail operation codes)

L (LOCATE SYMDEF)

The L command is limited to those programs and subprograms written in FORTRAN or those that utilize the standard GMAP SAVE macro to identify their entry point. It provides a means for locating an entry point (SYMDEF) location for one or more specified SYMDEF names. The technique employed involves a pattern search of memory to identify expansions of the SAVE macro. When an expansion is located, the address of the error linkage pair (.E.L..) can be determined. The second word of this pair contains the BCD name of the first SYMDEF defined in the program. If this name matches the requested name and other save expansions cannot be found referencing the same error linkage pair, the location of the first word of the save expansion is the desired SYMDEF address. Ambiguity exists if multiple SAVEs reference the same error linkage; i.e., the name specified in the error linkage cannot be identified with the proper SAVE. All names are listed, however.

Permissible forms of the command are:

Form	Meaning
L	Locate and list addresses of all SYMDEFs.
L S1,S2,...,Sn	Locate and list addresses of only the SYMDEFs specified.

The SYMDEF names and their corresponding absolute addresses are listed, provided they can be located. If a specified SYMDEF cannot be located, its absence in the listing indicates failure to find it.

A SAVE can be used to identify the entry SYMDEF for the main program, as well as for subprograms.

Example: L OP:N,CLOSE,PUT,SUBL

MA,MQ,ME,MI,MXn,MARn (MODIFY REGISTER)

These commands can be used to modify (change) the contents of a register.

Permissible forms of the commands are:

Form	Meaning
MA D1	Modify the A-register; i.e., replace its contents with D1.
MQ D1	Modify the Q-register.
ME A1	Modify the E-register. A1 is greater than 0 and less than 400.
MI A1	Modify the indicator register.
MXn A1	Modify index register n.
MARn D1	Modify address register n. D1 is an eight digit octal number

Examples: MA3271402/MQ0 (modify A and Q)

ME 377

MX4,400000 (note requirement of the delimiter)

O (ESTABLISH OFFSET)

The O command is used to establish an offset, or to change an existing one. When an offset is in effect, all communication between the user and the trace package concerning memory addresses are relative to the offset, unless otherwise specified. The only permissible forms of this command are:

Form	Meaning
O A1	Set offset value to A1.
O	Display current offset.

If this command is not used, all address references are absolute; i.e., an offset of zero will be in effect.

Example: 0110

OCT (OCTAL-TO-DECIMAL CONVERSION)

The OCT command converts an octal number to its decimal equivalent. The only permissible form of the command is:

Form	Meaning
OCT n	Convert the given octal number n to its decimal equivalent and display.

Example: OCT 777777

P OR PA (PATCH MEMORY)

The P command is used to patch a contiguous area of memory, starting at a specified address (A1). The patch data (D1,D2,...Dn) is processed serially and stored in ascending locations, starting at A1. Each such Di can assume one of the following forms:

Di - Patch the next location with Di.

n*Di - Patch the next n locations with Di.

RDi - Add the offset to the left half of Di (bits 0-17) before inserting the patch.

DiR - Add the offset to the right half of Di (bits 18-35) before inserting the patch.

RDiR - Add the offset to both halves of Di before inserting the patch.

The form, n*Di, can also be used when Di is prefixed and/or suffixed with the relocation flag (R).

Permissible forms of the command are:

Form	Meaning
P A1,D1,D2,...,Dn	Patch the specified data into memory starting at location offset+A1.
PA A1,D1,D2,...,Dn	Patch the specified data into memory starting at absolute location A1.

Examples: P573 314420623123,422046262066,513163314527

PA 3622 3*R712000000 102 50*0

R OR RA (RUN; I.E., RESUME EXECUTION)

The R command is used to resume execution of the target program. Unless the trace mechanism has been previously engaged, the trace package loses control until either a breakpoint is executed, or a TSX1 or XED TRACE invokes the package again. If execution is being resumed from a serviced breakpoint, the original instruction at the breakpoint location is executed at this time (provided a run address, A1, is not specified).

Permissible forms of the command are:

Form	Meaning
R	Resume execution.
R A1	Resume program execution at the effective address offset+A1.
RA A1	Resume program execution at the absolute address A1.

S,SA,SI,SIA (SNAP MEMORY)

These commands are used to snap, or display, a contiguous area of memory, starting at a specified address (A1). The snapshot is double-spaced and printed in the following format:

```
ADDRESS  DATA    DATA    DATA    DATA
```

If the SI or SIA form of the command is used, the corresponding mnemonic operation code and modifier are listed beneath each data word (provided the operation code is legal). In addition, the legal derail instructions show the service function name; e.g., FILACT, KOUT, etc.

The first line of a snapshot has the letter R or A appended to the address to indicate whether the address is relative to the offset or absolute.

Permissible forms of the S command are given below. The effective address for each form given is offset+A1. In all cases, SI can be substituted for S if mnemonic interpretations are desired. In addition, SA or SIA can be substituted for S if the specified starting address, A1, is absolute.

Form	Meaning
S A1	Snap location A1 only.
S A1,n	Snap n locations starting at A1.
S A1-A2	Snap the interval from A1 to A2, inclusive.

Snapshot lines which duplicate the last line printed are not shown. An asterisk (*) is appended to the address of the next line shown, if any, to indicate the omission.

Examples: <u>SA1/SA3/SA5</u> (snap absolute locations 1, 3 and 5)

<u>SI472,10</u> (snap and interpret 8 locations, starting at 472)

<u>S 100-200</u> (snap the interval from 100 through 200)

SAVE (SAVE CURRENT PROGRAM STATE)

The SAVE command is used to save the current state of the program. Upon receipt of the SAVE command, the entire contents of allocated memory is written to a temporary file (*TCP), created for this purpose by the trace package. The command can be exercised as often as desired. Parameters are not specified.

For a variety of reasons such as no file space available, or I/O error, the SAVE command can be unsuccessful in completing its function. Its success or failure is always reported to the user.

RESTORE (RESTORE PROGRAM STATE FROM LAST SAVE)

The RESTORE command is used with the SAVE command. It backs the program up to the point where the last SAVE command was issued. Its operation includes the following steps:

1. If the size of allocated memory has changed since the last SAVE took place, the size is adjusted accordingly.
2. The *TCP file is "bootstrapped" into memory.
3. The success or failure of the RESTORE command is reported to the user.
4. At this point, the entire state of the program has been restored and the trace package appears to have just completed a SAVE command. If other commands had been specified after the SAVE command on the same input line, they are now processed again.

Multiple restores from the same SAVE are permissible. All trace options, breakpoint locations, etc. in effect at the time of the SAVE are reinstated, even though they may have changed. Files in use at the time of the SAVE are not repositioned.

Multiple commands cannot be specified on the same input line following the RESTORE command. As with SAVE, parameters are not specified with the RESTORE command.

TERMINATE (TERMINATE EXECUTION VIA DRL RETURN)

The TERMINATE command is used to terminate execution and return to the level at which execution was invoked; i.e., build mode or system selection.

X,XA,XQ, X0,XE,XI,XB,Xn,ARn,AR (Display Register)

These commands are used to display the contents of all registers, or a selected register only.

Permissible forms of the command are:

Form	Meaning
X	Display the A,Q,E,I and all index registers.
XA	Display the A-register only.
XQ	Display the Q-register only.
XE	Display the E-register only.
XI	Display the indicator register only.
XB	Display the base address register.
X0	Display index register 0.
X1	Display index register 1.
X2	Display index register 2.
X3	Display index register 3.
X4	Display index register 4.
X5	Display index register 5.
X6	Display index register 6.
X7	Display index register 7.
ARn	Display address register n, where n is an octal digit between zero and seven.
AR	Display all address registers.

TRACING

Tracing, as applies to the following discussion, is a technique which consists of simulating the functional operation of a computer by utilizing one program to interpret and execute the instructions of another program. This technique provides debugging and instrumentation capabilities, since each instruction of the traced (target) program can be dynamically examined before it is executed. For example, any desired analysis can be made of the instruction's operation code, modifier, effective address, etc.

The purpose of the debug trace package is to provide a simulator or trace mechanism that is used in the time sharing environment. Seven different types of traces are available: TRANSFER TRACE, OPERATION CODE TRACE, MODIFIER TRACE, USE TRACE, CHANGE TRACE, FULL TRACE and MAP TRACE. An eighth type (OWN CODE TRACE) permits the user to gain control before each instruction is executed.

The command which causes the trace mechanism to become engaged is TRACE (or T). Upon receipt of this command, the trace package responds with the question,

TYPE?

This is a request to select the type of trace to be performed. The user must, at this time, enter one of the following:

TRA
OP
MOD
USE
CHG
FULL
MAP
OWN

If the selection is anything other than TRA, FULL or MAP, the trace package issues a request for parameters related to the selected type. For example, if OP was specified, the user is asked to enter the operation codes of the instructions that are to be traced.

The dialogue between user and trace package up to this point can be eliminated, if desired, by including the type selection and parameters (if required) on the same line as the trace command. For example, the following is a request to engage an operation code trace on the TSX1 and RET instructions:

```
Examples:  ?TRACE

           TYPE? OP

           ENTER OPS: TSX1,RET
```

This can be expressed on a single line as:

TRACE,OP,TSX1,RET

The only other information required before the trace mechanism can be engaged consists of the memory locations or intervals where tracing is to take place. The following message will be issued requesting this information:

ENTER TRACING REGION:

With the single exception of the map trace, the user can respond with any number of locations and/or intervals specifying the areas of memory he wishes to trace. An interval specification must take the form of A1-A2, which implies all locations from A1 to A2, inclusive. All locations and location intervals are considered relative to the offset, provided one has been previously established. To specify an absolute location or location interval, the specification may be suffixed with the letter A. For example, to trace (1) the absolute interval from 1310 through 1570, (2) relative location 4633, and (3) the relative interval from 4656 through 4700, the user responds:

ENTER TRACING REGION: 1310-1570A,4633,4656-4700

A null response to this request (a carriage return only), implies the interval specification, 0-777777A. Thus, tracing takes place throughout all allocated memory.

The tracing region specifications may also be included on the same line as the trace command, selection type and parameters by separating them with a slash (/). For example:

?TRACE,OP,TSX1,RET/1310-1570A,4633,4656-4700

or, if the default interval (0-777777A) is desired:

TRACE,OP,TSX1,RET/

By using this form, dialog between user and trace package can be completely eliminated. With the trace mechanism now engaged, a question mark (?) is issued indicating readiness to accept another command. Following the carriage return, unless the last command was a RUN or EXECUTE, the question MORE? is typed to indicate that another command is expected.

At the time the next RUN command is issued, the simulation process begins; i.e., every instruction of the user's program is executed in the controlled environment provided by the trace mechanism. This results in the following functions being performed:

1. The pseudo instruction counter (PIC) that is maintained by the trace mechanism is updated to the address of the next instruction to be executed. Unless the previous instruction was a repeat, EIS, derail or one that resulted in a transfer, the PIC is incremented by one.

2. Next, the instruction word at the address furnished by PIC is obtained and the operation code is examined. If the operation code is found to be illegal, the following message is issued to the user:

   NNNNNN: OP CODE FAULT RESULTING IF EXECUTED

   Where: NNNNNN is the address of the offending instruction, relative to the offset, if one has been previously established.

   As with all other error conditions detected by the trace mechanism, the user is immediately returned to command level. Since the trace mechanism never attempts to execute an instruction having illegal properties, the user has three alternatives to resume execution:

   a. Patch the offending instruction before issuing the next RUN command.

   b. Specify an address with the RUN command to bypass the instruction.

   c. Disengage the trace mechanism before issuing the next RUN command.

3. If the instruction is not a repeat, derail or character/byte store, its final effective address is determined. During this process, the instruction's modifier is examined to ensure it is legal. Also, if indirection is specified, the indirect chain is traversed and all modifiers encountered are similarly examined. Any of the following error messages can be issued as a result of this analysis:

   NNNNNN: ILLEGAL MODIFICATION RESULTING IF EXECUTED
   or
   NNNNNN: TAG FAULT RESULTING IF EXECUTED
   or
   NNNNNN: MEMORY FAULT RESULTING IF EXECUTED

   A memory fault error message is issued if the address of an indirect word is found to be out-of-bounds.

4. Having established the effective address, an analysis is now made to determine if the instruction requires a memory cycle or will result in a transfer of control. In both cases, the effective address is examined to ensure that it is within the boundaries of allocated memory. Furthermore, a test is made to determine if the effective address is referencing a location within the trace package. This analysis permits recognition of (1) a breakpoint location previously established by the B command, and (2) a TSX1, XED or STX1 addressing the trace package entry point (TRACE). If the instruction results in a store or transfer referencing some location in the trace package and is neither (1) nor (2), the following error message is issued:

   NNNNNN: STORE OR TRANSFER INTO TRACE PKG. RESULTING IF EXECUTED

5. If the PIC is within any of the specified tracing regions, the instruction is now ready to be examined according to the type of trace that has been selected. For example, if an Operation Code trace is engaged, the operation code of the instruction is compared with the operation code(s) selected by the user to be traced. Providing (1) all conditions are met, (2) a map trace is not engaged, and (3) trace output is not being queued, a single line of output is issued to the user indicating the following:

   a. Location of the instruction, relative to the offset, if one is established.

b. The mnemonic operation code.

c. The address field (bits 0-17) of the instruction word.

d. The modifier, if any.

e. The effective address, if applicable. This is also relative to the offset, provided one is established.

For example:

```
004633  TSX1  000502,IC  (005335)
```

EFFECTIVE ADDRESS

TAG FIELD

INSTRUCTION Y-FIELD

OPERATION CODE

LOCATION OF INSTRUCTION

If an offset is in effect and either the location or effective address is below the offset, the absolute address is displayed with an A appended to the appropriate field(s).

If the displayed instruction is one that is being executed by a XEC or XED, the location shown is that of the execute instruction, not the location of the instruction being executed with an X appended to the location field.

A somewhat different format is used to display a repeat-type instruction. In place of the instruction address, modifier and effective address fields, the repeat count, increment and terminate condition(s) is shown. Repeat count and increment are both given in octal. The repeated instruction(s) is also displayed on subsequent lines. These can be identified by the absence of the location field; i.e., this field is blank.

6. The display that results from a multiword instruction is somewhat different. For example, the following instruction:

```
MVT   (,,1),(1,1),20,1

ARG   DESC1

ADSC9 ASCTR,1,X7

ARG   TABLE
```

would result in the following:

```
000706  MVT  (000200  0  6BCD)(000400  1  4ASC)(002000 )
                                                └ADDRESS OF "TABLE"
                                          └NINE-BIT  ASCII
                                        └FOUR CHARACTERS
                                     └STARTING CHAR. POSITION
                                        ONE
                              └ADDRESS OF "ASCTR"
                         └SIX-BIT BCD
                        └SIX CHARACTERS
                     └STARTING CHAR. POSITION
              └ADDRESS OF "BCDSTR"
        └OPERATION CODE
└LOCATION OF INSTRUCTION
```

The first argument of the MVT means the first descriptor is indirect; i.e., the actual descriptor is at location DESC1. At DESC1, the descriptor is ADSC6 BCDSTR,0,6. The second argument indicates that the operand length is contained in a register, in this case X7. The remaining arguments of the MVT are the fill character and the truncation fault enable.

The descriptor types are as follows:

ADSC4	PKD
ADSC6	BCD
ADSC9	ASC
ARG	(none)
BDSC	BIT
NDSC4	PKD
NDSC9	ASC

7. At this point, the instruction is ready to be executed or, in the case of certain instructions, simulated. Simulation is necessary if the instruction (1) results in a transfer of control, (2) stores the instruction counter (IC) register, or (3) is an XEC or XED. Otherwise, all of the user's registers are restored and the instruction (or a tailored version of it) is executed. Tailoring, if applicable, is performed during step three and consists of stripping off modifiers of the following types:

All RI and IR types

IC, IDC and DIC

Derail instructions require special consideration by the trace mechanism before they can be executed. In the time sharing environment, a derail is used to request a service function to be performed by the Time Sharing Executive, and is analogous to the master mode entry used in the batch environment. The type of request, identified by the address portion of the derail, is first validated by the trace mechanism. If found to be illegal or privileged, the following error message is issued:

NNNNNN: ILLEGA' DRL RESULTING IF EXECUTED

Next, a test is made to determine if the derail is attempting to terminate execution. If so, the following message is issued:

NNNNNN: TERMINATION VIA DRL XXXXXX RESULTING IF EXECUTED

where: XXXXXX is the symbolic name of the terminating service function.

This causes ABORT, DRLDSC, RETURN or SYSRET. Also, under certain conditions, the service functions RELMEM and RESTOR can cause issuance of this message.

Unless the derail is a KIN request, its calling sequence is now reconstructed within the trace mechanism, tailored if necessary, and executed. Tailoring, in this sense, consists of changing the return address for service functions such as ADDMEM, RELMEM and RESTOR so that the trace mechanism does not lose control when the derail is executed.

If the derail is a KOUTN (Keyboard Output then Input), the trace mechanism immediately follows its execution with a KIN (Keyboard Input) to obtain the user's response. Upon encountering a subsequent DRL KIN in the target program, the trace mechanism simulates it by moving the input received from the last KOUTN to the user's buffer. This procedure is necessary because the trace package can perform keyboard I/O between the user's KOUTN and KIN, thus destroying his actual input data.

The KIN simulation requirement is a means to initiate a breakpoint and enter command level any time the target program does a KOUTN. This can be accomplished by responding to the program's input request with the following:

$BRK

Upon intercepting the user's input response and identifying this signal, the trace mechanism issues the following message:

ENTER ACTUAL INPUT DATA:

At this time, the user must enter the actual data required by his program. Following this, the trace package enters command level and the user is now free to exercise any of the commands available. Traced execution resumes at the time the next RUN command is issued.

TYPES OF TRACES

The eight types of traces available to the user are described below.

Transfer Trace (TRA)

The transfer trace displays every instruction that results in a transfer of control, provided the instruction causing the transfer is in one of the specified tracing regions. A conditional transfer TZE, etc. is displayed only when it actually transfers.

Operation Code Trace (OP)

An operation code (op code) trace can be used to display every occurrence of the use of one or more selected operation codes, provided the instruction using the operation code is in one of the specified tracing regions. When this selection is given in response to TYPE?, the user is asked to enter the operation codes he wishes to trace.

ENTER OPS:

At this time, the user must respond with one or more mnemonic operation codes. A blank or comma is used to separate multiple specifications. If the user desires to trace the DRL operation code, he can specify a list of only those derail service functions he wishes to trace. This list, parenthetically enclosed, must immediately follow the derail operation code specification with no intervening blank or comma. The list can consist of one or more service function names and/or equivalent octal values. For example, if it is desired to trace all repeat-type instructions and the derail service functions DIO, FILSP and REW, the user responds:

ENTER OPS: RPT,RPL,RPD,DRL(DIO,FILSP,REW)

or,

ENTER OPS: RPT,RPL,RPD,DRL(1,13,12)

Modifier Trace (MOD)

The modifier trace is similiar to the operation code trace, except that the use of selected modifiers are traced instead of operation codes. The following message is issued requesting the modifiers to be traced:

ENTER MODS:

The response must consist of one or more modifiers specified in the same manner as for GMAP coding. Only first-level modifiers or single-word instructions are traced i.e., modifiers encountered in an indirect chain will not be considered. For example, the following response would trace all uses of index register 1 for modification purposes and the modifiers DL,SC and CI:

ENTER MODS: 1,1*,*1,DL,SC,CI

The modifiers N, N* and *N can also be specified; however, N* must be specified as simply *.

Where: N = register
N* or (*) = register, indirect
*N = indirect, register

Use Trace (USE)

The use trace displays every instruction that uses, or references, one or more specified registers and/or memory locations, provided the instruction is in one of the specified tracing regions. The following message is issued requesting the specifications:

ENTER REGS AND/OR LOCS TO TRACE:

Any combination of register designators, memory locations or memory location intervals can be given as a response. As with the tracing region specifications, a location or location interval may be suffixed with the letter A if it is given as absolute. In this case, the offset is not added to it.

Register specifications may include:

A - A-register
Q - Q-register
AQ - combined A and Q registers
E - exponent register
EAQ - combined E, A and Q registers
Xn - index register 0 through 7
I - the indicator register
ARn - address register 0 through 7

For example, if the user desired to trace (1) all uses of the combined AQ register, (2) references to relative location 1472, (3) all uses of index registers 1 and 4, and (4) references to any location in the absolute interval from 0 through 143 (slave prefix area), the response could be:

ENTER REGS AND/OR LOCS TO TRACE: AQ,1472,X1,X4,0-143A

In addition all traced EIS instructions can be delayed by inserting the specification "EIS"; e.g.,

```
T USE/AQ,1472,EIS,X1,X4,0-143A
```

All multiword EIS instructions are properly traced. However, certain types of memory location references are not detected by the trace mechanism for both use and change tracing. Only the effective address of an instruction referencing memory is tested for the use or change conditions. This precludes the detection of implicit multiple word references, as is the case with LREG, SREG and double-precision instructions. Other undetected references include:

1. Locations referenced by repeated instructions

2. Locations referenced by derail service functions

Change Trace (CHG)

The change trace is similiar to the use trace except that those instructions that actually change the specified register(s) or memory location(s) are displayed. A change trace is very useful for locating the cause of a register or location that is being unexpectedly destroyed.

The "EIS" specification for this trace displays EIS instructions that alter registers.

Full Trace (FULL)

A full trace displays every instruction executed by the program, provided the instruction is in one of the specified tracing regions. The volume of output received from this trace is prohibitive for a large tracing region. However, if output is queued in lieu of being immediately issued, a full trace can be of much value in revealing the final steps that lead up to an unexpected error condition. Output queuing is discussed in Queue Trace Output in this section.

Map Trace (MAP)

The map trace is used for both debugging and instrumentation purposes. Instrumentation, as used in this context, implies measurement of a program's efficiency. By engaging a map trace in the program and allowing the traced program to run to completion, the user can then display a map report showing partitioned segments of memory and the number of instructions that were executed in each such segment. Thus, heavily used areas of code can be identified and appropriate action can be taken to optimize these areas.

This is the only type of trace which requires the tracing region specification to consist of a single interval. A null response to the tracing region request implicitly defines the absolute interval, 0-777777.

The trace package is assembled with a 512 (decimal) word area dedicated for mapping. Depending on the size of the specified tracing region interval, each word of this dedicated area corresponds to some subdivided interval of the tracing region. Whenever the trace mechanism executes an instruction that is in the tracing region, it determines the subdivision the instruction is in and increments the corresponding word in the map area by one. Repeat-type instructions are incremented according to the number of times the repeated instruction(s) are executed. It is possible for the repeat and the instruction(s) influenced by it to be in different subdivisions.

The size of each subdivision is a function of the tracing region specification and is determined as follows:

1. The total number of locations (N) implied by the tracing region interval (A1-A2) is computed; e.g., N=A2-A1+1.

2. A trial subdivision size is now determined. It is the integral part of the quotient resulting from the division, (N+511)/512.

3. If the trial subdivision size is an even power of 2 (e.g., 1,2,4,8,...), then this is the size that is used; otherwise, the first power of 2 that is greater than the trial size is used.

It generally takes at least two map trace runs to identify the actual code that is monopolizing the processor. Suspect areas can be isolated by first using a relatively large tracing region specification. After displaying the map for this run, the user can then select the subdivision interval having the most activity and run the program again using this interval as the tracing region. The second run provides a finer resolution. The SAVE and RESTORE commands are a means of making repeated runs.

Own Code Trace (OWN)

The own code trace permits the user to gain control from the trace mechanism before each instruction is executed (or simulated). To engage an own code trace, the user must supply a special subroutine to perform his desired tracing function(s). The entry point to this subroutine is requested by the trace package, as follows:

ENTER ABS. ADDRESS OF OWN PROCEDURE:

At this time, the user must enter the absolute address of the entry point to his subroutine procedure. When the next RUN command is issued, control is passed to this entry point via a TSX1 before each instruction is executed, provided the instruction is in one of the specified tracing regions. This includes every step of an execute (XEC or XED) chain, but does not include repeated instructions; i.e., the subroutine is entered before the repeat instruction is executed, but not before each execution of the repeated instruction(s). Upon each entry to the subroutine, the following registers are set by the trace mechanism:

X0 - contains a pointer to the target program's registers. These registers are loaded and stored by LREG and SREG instructions addressing this area.

X1 - the linkage register between the trace mechanism and the user's subroutine.

X3 - contains the pseudo instruction counter (PIC). The PIC reflects the location of the current instruction being executed.

X4 - contains a pointer to the location of the instruction word. Except for instructions being executed via XEC or XED, X3 and X4 are identical.

X5 - index 5 contains the effective address of the single word instruction, if applicable.

IR - loaded with the target program's indicators.

X6 - nonzero if instruction is multiword EIS.

QR - individual bits in this register reflect attributes of the instruction, as follows:

Bit No.	Meaning If Bit Is On
00	Unconditional transfer
01	Conditional transfer
02	RPT, RPL or RPD
03	XEC or XED
04	STCA, STCQ, STBA OR STBQ
05	STC1, STC2 or TSXn
06	Instruction changes the register(s) Specified by bits 18-33
07	Instruction is a store-type
08-16	Reserved
17	DRL
18	Set if EIS is executing
19	If OP being traced
20	Memory cycle(s) required
21	DR Tag field (EIS)
22	AR utilized
23	QR utilized
24	AQ utilized
25	ER utilized
26	EAQ utilized
27	IR utilized
28	X0 utilized (AR0 for EIS)
29	X1 utilized (AR1 for EIS)
30	X2 utilized (AR2 for EIS)
31	X3 utilized (AR3 for EIS)
32	X4 utilized (AR4 for EIS)
33	X5 utilized (AR5 for EIS)
34	X6 utilized (AR6 for EIS)
35	X7 utilized (AR7 for EIS)

Following analysis, the user must return to the trace mechanism so that the instruction can be executed and the simulation process continued. With the linkage register (X1) restored to its contents at entry, the user has two alternatives to relinquish control:

1. TRA 0,1 - This return prevents the instruction from being displayed.

2. TRA 1,1 - This return causes the instruction to be displayed. If pause mode is enabled, command level is entered before the instruction is executed.

Registers other than X1 need not be saved or restored by the user's subroutine. They are properly restored by the trace mechanism when the subroutine returns.

When using the own code trace, the user must ensure that his trace subroutine does not use code that is also used by the target program or uses shared data areas.

If the SAVE macro is used to identify the entry point to the user's trace subroutine, the L command can be used to locate its absolute address in memory.

COMMANDS RELATED TO TRACING

DISPLAY (Display Trace-Related Information)

The DISPLAY command is used to selectively display information that is collected by the trace mechanism. The command must always be accompanied with a literal parameter indicating what is to be displayed. This literal can assume any of the following forms:

DRLS - This selection provides a listing of derail service function usage frequencies. The symbolic name of each service function used by the target program is displayed, together with a decimal value indicating the number of times the service function has been used. This frequency table is maintained by the trace mechanism for all types of traces and is updated without regard to the tracing region specifications. While all frequencies are preset to zero, the table is never implicitly initialized by the trace package. If desired, the I command can be used to initialize it.

MAP - If a map trace is engaged, this selection displays the resulting map report showing consecutive subdivisions of the tracing region interval and the number of instructions executed in each subdivision. Columns of this report are identified by the heading title:

```
LOC. INTERVAL    ID   FREQUENCY    %     CUM %
```

or, if the subdivision size is unity,

```
LOC.     ID   FREQUENCY     %     CUM %
```

LOC. INTERVAL represents the range of the subdivision and consists of an address interval, A1-A2. This interval implies all locations from A1 to A2, inclusive. The first line of the report has an A or R appended to the interval specification indicating whether the addresses are absolute or relative. The meaning of other column identifiers are:

ID - This column contains the mnemonic operation code of the instruction word at location A1; i.e., the first location of the subdivision. If the operation code is illegal, this field is blank.

FREQUENCY - This column contains a decimal count indicating the total number of instructions that have been executed in the subdivision. Subdivision intervals that have had no activity (the frequency is zero) are not shown in the report.

%   - This column contains a decimal quantity indicating what percent the subdivision's frequency is of the total number of instructions executed by the target program.

CUM %   - This column indicates cumulative percentage. It is a running total of the % column.

Both percentages mentioned above are rounded to the nearest hundredth of a percent; e.g., 23.084 would be truncated to 23.08, while 23.085 would be rounded up to 23.09. Percentages are not shown for subdivisions having a frequency less than .005% of the total number of instructions executed by the target program.

If desired, a partial map report can be displayed, showing only subdivisions with the highest frequencies. This optional form of the DISPLAY is requested by accompanying the map selection with an octal value (N), indicating how many subdivisions are to be shown. The resulting map report consists of the N most active subdivisions sorted in descending order according to frequencies.

M#   - This selection displays the total number of instructions executed in the mapped region; i.e., the sum of all subdivisions. The value is given in decimal.

Q   - If the Q command has been used to queue trace output (in lieu of issuing it to the user's terminal), this display selection shows the 25 most recent lines of trace output that would have otherwise been printed. An octal numeric greater than 0 and less than 32 can optionally accompany the Q display selection. If present, only the last N accumulated output lines are displayed. In all cases, output is shown in the proper sequence.

Q#   - This display selection shows the total number of entries that have been made in the output queue. The value is given in decimal.

T   - Whenever the trace mechanism encounters an instruction that results in a transfer of control, an entry is made in a special transfer queue that is structured similiar to the output queue. This queue is maintained for all types of traces and entries are made without regard to the tracing region specification. If an unexpected error condition occurs, the steps leading up to the error can be viewed by displaying the transfer queue. As with the output queue, the last 25 entries are shown, unless an octal numeric accompanies the selection. In this case, only the last specified number of transfers are shown. This feature provides an identical situation of a transfer trace with queued output in the tracing region, 0-777777A.

T#   - This selection shows the total decimal number of entries that have been made in the transfer queue.

- This selection shows the total decimal number of instructions that have been executed (or simulated) by the trace mechanism. The value is the iteration count for repeated instructions, as well as the instructions executed by XEC/XED chains. The map display selection utilizes this value for percentage calculations.

*   - This selection displays the next instruction of the target program to be executed.

XF   - This selection displays the address registers, unless they are all zeros. It always displays the A, Q, E, I and all index registers.

Permissible forms of the DISPLAY command are:

Form	Meaning
DISPLAY DRLS	Show the number of times each derail service function has been used.
DISPLAY MAP	Display frequencies for consecutive subdivisions of the tracing region which has been mapped.
DISPLAY MAP N	Display the first N subdivisions having the highest frequencies.
DISPLAY M#	Display the total number of instructions executed in the mapped region.
DISPLAY Q	Display the last 25 lines of trace output which have been queued.
DISPLAY Q N	Display the last N lines of trace output which have been queued.
DISPLAY Q#	Display the total number of output lines which have been queued.
DISPLAY T	Display the last 25 instructions which caused a transfer of control.
DISPLAY T N	Display the last N instructions which caused a transfer of control.
DISPLAY T#	Display the total number of entries made in the transfer queue.
DISPLAY #	Display the total number of target program instructions executed.
DISPLAY *	Display the next instruction of the target program to be executed.

I (Initialize Queues/Frequencies)

The I command can be used to selectively (or collectively) initialize queues and frequency counters that are maintained by the trace mechanism. As with the DISPLAY command, a literal parameter must be specified, provided selective initialization is desired. This parameter can assume any of the following forms:

DRLS - This selection resets all derail service function usage frequencies to zero. The I command is the only means by which the derail frequencies can be initialized.

MAP - This selection initializes the internal mapping area; i.e., all subdivision frequencies are set to zero. This initialization is always implicitly performed whenever the TRACE command is utilized to engage a map trace.

Q - This selection initializes the trace output queue, provided the Q command has been previously issued to enable output queuing.

T - This selection initializes the trace transfer queue. The transfer queue is always implicitly initialized whenever the TRACE command is issued to engage the trace mechanism.

- This selection resets the count of the number of instructions executed (or simulated) by the trace mechanism. As with the T selection, implicit initialization of this count occurs whenever the TRACE command is issued.

Permissible forms of the I command are:

Form	Meaning
I	Initialize DRLS, MAP, Q, T and #.
I DRLS	Initialize derail service function usage frequencies.
I MAP	Reset all subdivision frequencies in the mapping area to zero.
I Q	Initialize the trace output queue.
I T	Initialize the transfer queue.
I #	Reset the count of the total number of instructions which have been executed by the trace mechanism.

NOTRACE (Disengage Trace Mechanism)

The NOTRACE command, which may be given as N, causes the trace mechanism to become disengaged. The program becomes "free running" when the next RUN command is issued.

PAUSE (Pause Before Instruction Execution)

The PAUSE command places the trace mechanism in a step mode. When this mode is enabled, the user is returned to command level before each instruction that meets the tracing conditions is executed. After printing the instruction, a question mark (?) is issued indicating readiness to accept a command. When the next RUN command is given, the instruction is executed and tracing continues until the next instruction meeting the conditions is encountered. The E command can be used if it is desired to execute the instruction and immediately return to command level.

If the next instruction to execute is to be patched while in the PAUSE mode, it is necessary to perform the patch and issue the command:

R A1

Where: A1 is the address of the patched instruction.

Certain conditions can occur which prevent the trace mechanism from being able to enter the command level before executing an instruction. When such a condition exists, it is indicated on the output line; i.e., the following message is appended to the line:

...CAN'T PAUSE

The various conditions that prevent the capability to pause are:

1. Execution of an instruction that has been replaced by a breakpoint.

2. Execution of the instruction(s) influenced by an XEC or XED.

3. Execution of an instruction having IDC or DIC modification.

4. Execution of a TOV, TEO or TEU instruction which results in a transfer.

NOPAUSE (Discontinue Pause Mode)

The NOPAUSE command is the only means of cancelling the effect of a prior PAUSE command. The mode is never implicitly cancelled, even though a different type of trace may be engaged.

Q (Queue Trace Output)

This command enables queuing of trace output, in lieu of immediately issuing it to the terminal. When this mode is in effect, the last (most recent) 25 lines of output that would have otherwise been issued to the terminal are saved. The queued output can, at any time, be displayed by exercising the DISPLAY command.

Output queuing can be used in a variety of ways. For example, the steps leading up to an unexplained error condition can be revealed by engaging a Full trace with queued output. When the error occurs, a display of the queue usually shows what has happened. Another useful application consists of determining the number of times a certain operation code or modifier has been used by simply engaging the appropriate type of trace and queueing the output. The number of entries made in the output queue can then subsequently be displayed, and this value reflects the number of times the selected operation code or modifier was used.

The Q command must be given after the trace mechanism has been engaged. Its effect is cancelled only if another trace command is subsequently given.

ERROR MESSAGES

1. ILLEGAL INPUT--RETYPE

   This error message is issued if (1) an unknown command is given, (2) a syntactical error is discovered while processing a command, or (3) the command cannot, for some reason, be performed. If multiple commands are specified on the same input line, any error detected with a command causes all subsequent commands on that line to be ignored.

2. ILLEGAL INPUT--TRACE NOT ENGAGED

An unknown type of trace, illegal parameters or an invalid tracing region has been specified with the TRACE command. The command must be reissued properly before the trace mechanism is engaged.

3. TABLE SPACE EXHAUSTED--TRACE NOT ENGAGED

A use or change trace has specified too many locations and/or location intervals to trace, or the number of locations and/or location intervals constituting the tracing region is excessive. An aggregate area of approximately 128 words is used to store these locations and intervals. Each location specification requires two words of this area, while a location interval requires four words. For example, a total of up to 64 location specifications or 32 intervals could be accommodated.

4. ROOM FOR BREAKPOINT ENTRIES EXHAUSTED

This message is issued when no more breakpoints can be accepted. The user must delete at least one previously established breakpoint before a new one can be accepted. A maximum of 20 breakpoints can be concurrently active.

5. ILLEGAL TO REPLACE

A specified breakpoint location has either been found to be under the influence of a repeat-type instruction, or contains an illegal operation code.

6. UNKNOWN BREAKPOINT ENCOUNTERED

A breakpoint location has been executed sending control into the trace package; however, no record of ever having established the breakpoint can be found. More than likely, the target program has moved an area of memory containing a breakpoint location and is now executing it. The trace package assumes the replaced instruction was a NOP.

SUPPLEMENTAL INFORMATION

1. The total memory requirement of the trace package is approximately 20000 (octal) locations, or 8K.

2. In the Time Sharing FORTRAN environment, the L (locate SYMDEF) command provides load origins of subprograms and library modules, as well as SYMDEF addresses.

3. Although the trace package is implemented in floatable code and can be loaded anywhere in memory without relocation, the code cannot be moved once the package has been invoked.

4. Two locations within the trace package may be snapped to determine its size and date of assembly. These locations are relative to the entry SYMDEF (TRACE), as follows:

TRACE-3	Contains size in bits 0-17
TRACE-2	Contains BCD date of assembly, MMDDYY

5. The KOTNOW derail service function is utilized for issuing trace output to the terminal. If the break key is pressed while such a line is being typed, the trace mechanism does not recognize the interrupt until after it has executed the associated instruction.

EXAMPLE OF USAGE

The following example illustrates the use of the trace package. The target program consists of a GMAP-coded subsystem to create temporary files. In the dialog that follows, user responses are underscored.

```
USER ID-JOEDOE
PASSWORD--
XXXXXXXXXXXX
11 BLOCKS FILE SPACE AVAILABLE

*OLD DEFILSRC
*DISP:N


01/21/75     15.000


0010     $         IDENT     ACCNT,J.DOE
0020     $         LOWLOAD   36
0030     $         OPTION    NOGO,NOSETU,SAVE/DEFIL
0040     $         GMAP      NDECK
0055               REM
0060               REM       * * * * * * * * * * * * * * * * * * * * * * *
0070               REM       *                                           *
0080               REM       *   THIS PROGRAM CREATES TEMPORARY FILES.   *
0090               REM       *   PROVISION IS MADE TO SPECIFY (1) FILE   *
0100               REM       *   NAME, (2) MODE--LINKED OR RANDOM, AND   *
0110               REM       *   (3)  SIZE IN LINKS FOR EACH SUCH FILE   *
0120               REM       *   TO BE CREATED. A NULL RESPONSE TO THE   *
0130               REM       *   PROGRAM'S INPUT REQUEST FOR THIS DATA   *
0140               REM       *   WILL RESULT IN IMMEDIATE TERMINATION.   *
0150               REM       *                                           *
0160               REM       * * * * * * * * * * * * * * * * * * * * * * *
0170               REM
0180               LODM      .G3TSM    LOAD TSS MACROS,
0190               .SSDRL              AND DEFINE DRL NAMES.
0200               DETAIL    OFF
0210     ....      SAVE                ***PROGRAM ENTRY POINT.
0220               CALL      TRACE     INVOKE TRACE PACKAGE.
0230     NXTFIL    NULL                RE-ENTRY TO CREATE NEXT FILE.
0240               LDAQ      QUERY     PRESERVE O/P TALLY WORDS IN AQ.
0250               DRL       KOUTN     ISSUE REQUEST FOR PARAMETERS.
0260               ZERO      QUERY
0270               STAQ      QUERY     RESTORE TALLIES TO ORIGINAL STATE.
0280               DRL       KIN       OBTAIN USER'S RESPONSE.
0290               ZERO      INPUT,COUNT
0300               ZERO      STAT
0310               LDA       COUNT     NO. OF CHARACTERS HE TYPED.
0320               CMPA      1,DL      IF ONLY 1, IT HAS TO
0330               TN2       *+2       BE A CARRIAGE RETURN,
```

```
0330            TNZ     *+2         BE A CARRIAGE RETURN,
0340            DRL     RETURN       SO TERMINATE EXECUTION.
0350            ALS     6           OTHERWISE, PREPARE A
0360            ORA     TALLYB       TALLYB WORD FOR SCANNING
0370            STA     SCAN         THE INPUT LINE.
0374            LDA     1,DL        IN CASE NO MODE AND/OR #LINKS
0376            STA     NAME+2       GIVEN, ASSUME 1 LINK SEQUENTIAL.
0380            EAA     NAME
0390            TSX1    FIELD       GO GET THE FILE NAME.
0400            TRA     READY+1     RETURN 1--CR ENCOUNTERED IN SCAN.
0410            EAA     TEMP        RETURN 2--FIELD TERMINATED BY COMMA.
0420            TSX1    FIELD       NOW GET THE MODE.
0430            STC2    BYPAS       REMEMBER CR, IF WE RETURN HERE.
0440            LDA     TEMP        IF MODE FIELD IS NULL,
0450            CMPA    BLANKS       THEN ASSUME USER WANTS
0460            TZE     BYPAS        TO CREATE A LINKED FILE.
0470            ARL     27          OTHERWISE, ISOLATE 1ST CHARACTER.
0480            CMPA    LCL,DL      IS IT "L"?
0490            TZE     BYPAS       YES.
0500            CMPA    LCR,DL      NO, HOW ABOUT "R"?
0510            TNZ     ERROR       GO COMPLAIN IF IT'S NEITHER ONE.
0520            LDA     =1B18,DL    CHANGE MODE
0530            ORSA    NAME+2       TO RANDOM.
0540   BYPAS    EAQ     **          WAS #LINKS SPECIFIED?
0550            TNZ     READY       NO, ASSUME 1 LINK DESIRED.
0560            LDA     SCAN,SC     RE-ENTRY TO DEVELOP BIN. EQUIVALENT
0570            REM                  OF #LINK SPECIFICATION IN QR.
0580            TTF     NOTCR       TRA IF NOT CR.
0590            CMPQ    512,DL      IS SPECIFICATION REASONABLE?
0600            TRC     ERROR       NO, COMPLAIN.
0610            STBQ    NAME+2,04 YES, STUFF IT IN DEFIL ARG LIST,
0620            TRA     READY+1      AND GO CREATE THE FILE NOW.
0630   NOTCR    CMPA    =0040,DL    IF CHARACTER IS A BLANK,
0640            TZE     BYPAS+2      IGNORE IT.
0650            SBLA    =0060,DL    STRIP OFF ASCII ZONES
0660            CMPA    10,DL        AND MAKE SURE IT'S NUMERIC.
0670            TRC     ERROR
0680            STA     TEMP
0690            MPY     10,DL
0700            ADLQ    TEMP
0710            TRA     BYPAS+2     GET NEXT DIGIT, IF ANY.
0720   *
0730   READY    ERSQ    BYPAS       RESET CR FLAG FOR NEXT TIME.
0740            DRL     DEFIL       ATTEMPT TO CREATE THE FILE.
0750            ZERO    NAME,STAT
0760            LDA     SUCC        ASSUME WE WERE SUCCESSFUL.
0770            LDQ     STAT        CORRECT ASSUMPTION?
0780            TZE     *+4         YES.
0790            ORQ     =0060,DL    NO, SHOW USER THE STATUS WE GOT.
0800            STBQ    INFORM+5,04
0810            LDA     NSUCC
0820            STA     TEMP+2
0830            LDAQ    ITALS
0840            TRA     ERROR+1     GO INDICATE OUR SUCCESS OR FAILURE.
0850   *
0860   FIELD    NULL    **         *SUBR. TO GET NEXT INPUT FIELD.
0870            ORA     =01140,DL FWA OF RECEIVING AREA IN AR.
0880            STA     TEMP+2      TALLYB FWA,9
0890            LDAQ    BLANKS      INITIALIZE RECEIVING
0900            STAQ    TEMP+2,I     AREA WITH BLANKS.
0910   NXTCHR   LDA     SCAN,SC     RE-ENTRY TO GET NEXT CHAR. OF FIELD.
0920            TTF     *+2         IF IT'S A CARRIAGE RETURN,
0930            TRA     0,1          EXIT VIA 0,1.
0940            CMPA    COMMA,DL    IF A COMMA,
0950            TZE     1,1          EXIT VIA 1,1
0954            CMPA    =0040,DL    IF A BLANK,
0956            TZE     NXTCHR       IGNORE IT.
```

```
               CMPA      UCA,DL     FORCE
               TNC       *+4         UPPER CASE
               CMPA      UCZ+1,DL    ALPHABETICS
               TRC       *+2         TO
               ORA       =0040,DL    LOWER CASE.
               STA       TEMP+2,SC  PUT AWAY THE CHARACTER
               TTF       NXTCHR      AND GO GET THE NEXT ONE.
      ERROR    LDAQ      ETALS       ERROR. FIELD EXCEEDS 8 CHARACTERS.
               STAQ      TEMP
               DRL       KOUT
               ZERO      TEMP
               TRA       NXTFIL
      *
      *        N O N - P R O C E D U R E . . . . . . .
      *
      BLANKS EASCII      2,
      COMMA    BOOL      054        ASCII COMMA.
      COUNT    ZERO
      ETALS   ETALLY     TEMP+1,1
               TALLYB    *+1,17
               OCT       015012177177         CR,LF,RO,RO
               ASCII     4,INVALID INPUT
      INFORM   ASCII     6,UNSUCCESSFUL -- STATUS N
      INPUT    BSS       20
      ITALS   ETALLY     TEMP+1,2
               TALLYB    ETALS+2,4
      LCL      BOOL      154        ASCII LOWER CASE "L".
      LCR      BOOL      162        ASCII LOWER CASE "R".
      NAME    EASCII     2, *NAME*
               ZERO
      NSUCC    TALLYB    INFORM,24
      QUERY   ETALLY     *+1,1
               TALLYB    *+1,40
               OCT       015012177177
               ASCII     9,ENTER FILENAME,MODE(L OR R),#LINKS:
      SCAN     TALLYB    **,**
      STAT     BSS       2
      SUCC     TALLYB    INFORM,10,2
      TALLYB   TALLYB    INPUT,**
      TEMP    EBSS       3
      UCA      BOOL      101        ASCII UPPER CASE "A".
      UCZ      BOOL      132        ASCII UPPER CASE "Z".
               END
      $        EXECUTE
      $        LIMITS    1,7K,,500
      $        PRMFL     H*,W,R,HANSEN/DEFIL
      $        ENDJOB
*JRN:N,J,IDENT(A,JDOE)
SNUMB #5927T


*LODX DEFIL
PATCH,SAVE OR RUN? R
000152: FUNCTION? L
```

```
 .......-000146    TRACE-000417
?0146/SAVE
SUCCESSFUL
?F0 6002000
000071
?SI71,10

000071R 000006002000    000352000373    000375235000    000373236000
        DRL   DEFIL                     LDA             LDQ

000075  000247600000    000060276007    000322552004    000355235000
        TZE             ORQ      ,DL    STBQ            LDA
?B71
?TRACE
TYPE? FULL
ENTER TRACING REGION:   (NULL RESPONSE GIVEN)
?Q/R
ENTER FILENAME,MODE(L OR R),#LINKS: TMPFIL,L,2
000071: BREAKPOINT
?DISPLAY *
000071  DRL    DEFIL
?E
000073: FUNCTION? CALL STATF


LIST OF OPEN FILES: DEFILSRC TMPFIL1

CALL COMPLETED
?D/R
SUCCESSFUL
ENTER FILENAME,MODE(L OR R),#LINKS: $BRK
ENTER ACTUAL INPUT DATA:    (NULL RESPONSE GIVEN)
000012: FUNCTION? DISP Q14
000056  TRA   000237
000073  LDA   000375
000074  LDQ   000373
000075  TZE   000247
000101  STA   000402
000102  LDAQ  000350
000103  TRA   000275
000127  STAQ  000400
000130  DRL   KOUT
000132  TRA   000155
000007  LDAQ  000356
000010  DRL   KOUTN
?DISP Q#/DISP T#/DISP #
171
32
171
?R
000021: TERMINATION VIA DRL RETURN RESULTING IF EXECUTED
?RESTORE
SUCCESSFUL
?T MAP/0-242
?R
ENTER FILENAME,MODE(L OR R),#LINKS: TMPFIL2
SUCCESSFUL
ENTER FILENAME,MODE(L OR R),#LINKS: TMPFIL3 , RANDOM , 10
SUCCESSFUL
ENTER FILENAME,MODE(L OR R),#LINKS: $BRK
ENTER ACTUAL INPUT DATA: TMPFIL4,,999999
000012: FUNCTION? DISPLAY MAP
```

LOC.	ID	FREQUENCY	%	CUM %
000005R	TRA	1	.25	.25
000007	LDAQ	3	.76	1.02
000010	DRL	3	.76	1.78
000012	STAQ	2	.51	2.29
000013	DRL	2	.51	2.80
000016	LDA	2	.51	3.31
000017	CMPA	2	.51	3.82
000020	TNZ	2	.51	4.33
000022	ALS	2	.51	4.83
000023	ORA	2	.51	5.34
000024	STA	2	.51	5.85
000025	LDA	2	.51	6.36
000026	STA	2	.51	6.87
000027	EAA	2	.51	7.38
000030	TSX1	2	.51	7.89
000031	TRA	1	.25	8.14
000032	EAA	1	.25	8.40
000033	TSX1	1	.25	8.65
000035	LDA	1	.25	8.91
000036	CMPA	1	.25	9.16
000037	TZE	1	.25	9.41
000040	ARL	1	.25	9.67
000041	CMPA	1	.25	9.92
000042	TZE	1	.25	10.18
000043	CMPA	1	.25	10.43
000044	TNZ	1	.25	10.69
000045	LDA	1	.25	10.94
000046	ORSA	1	.25	11.20
000047	EAQ	1	.25	11.45
000050	TNZ	1	.25	11.70
000051	LDA	4	1.02	12.72
000052	TTF	4	1.02	13.74

000053	CMPQ	1	.25	13.99
000054	TRC	1	.25	14.25
000055	STBQ	1	.25	14.50
000056	TRA	1	.25	14.76
000057	CMPA	3	.76	15.52
000060	TZE	3	.76	16.28
000061	SBLA	2	.51	16.79
000062	CMPA	2	.51	17.30
000063	TRC	2	.51	17.81
000064	STA	2	.51	18.32
000065	MPY	2	.51	18.83
000066	ADLQ	2	.51	19.34
000067	TRA	2	.51	19.85
000071	DRL	2	.51	20.36
000073	LDA	2	.51	20.87
000074	LDQ	2	.51	21.37
000075	TZE	2	.51	21.88
000101	STA	2	.51	22.39
000102	LDAQ	2	.51	22.90
000103	TRA	2	.51	23.41
000104	ORA	3	.76	24.17
000105	STA	3	.76	24.94
000106	LDAQ	3	.76	25.70
000107	STAQ	3	.76	26.46
000110	LDA	26	6.62	33.08
000111	TTF	26	6.62	39.69
000112	TRA	1	.25	39.95
000113	CMPA	25	6.36	46.31
000114	TZE	25	6.36	52.67
000115	CMPA	23	5.85	58.52
000116	TZE	23	5.85	64.38
000117	CMPA	20	5.09	69.47
000120	TNC	20	5.09	74.55

```
000121     CMPA      18          4.58        79.13

000122      TRC      18          4.58        83.72

000123      ORA      18          4.58        88.30

000124      STA      20          5.09        93.38

000125      TTF      20          5.09        98.47

000127     STAQ       2           .51        98.98

000130      DRL       2           .51        99.49

000132      TRA       2           .51        00.00
?DISP #
393
?T OP DRL(KOUTN,KIN) /
?PAUSE
?R
000013  DRL    KIN
?R
INVALID INPUT
000010  DRL   KOUTN
?R
ENTER FILENAME,MODE(L OR R),#LINKS:    (NULL RESPONSE GIVEN)
000013  DRL    KIN
?TERM
SYSTEM ?BYE
   3 TEMPORARY FILES CREATED.

TMPFIL1  ?   (NULL RESPONSE GIVEN)
TMPFIL2  ?   (NULL RESPONSE GIVEN)
TMPFIL3  ?   (NULL RESPONSE GIVEN)
**COST:  $    0.17  TO DATE:  $   206.11 = 21%
**ON AT  15.000 - OFF AT 15.016 ON 01/21/75
```

SECTION VIII

SUPPORT FACILITIES

TIME SHARING MEDIA CONVERSION PROGRAM

The Time Sharing Media Conversion Program (TSCONV) is a batch program that may be run either at the central computer site or through a remote batch (GRTS or NPS) terminal. In input mode the program generates a standard system format, time sharing ASCII file from a suitable card deck. In output mode the program produces a card deck from a time sharing ASCII file to save the file in card form.

Operational Description

The media conversion program performs the following two functions based upon user-supplied directives. User directives are supplied at the first record images on the input file (I*).

- INPUT - create a standard system format, time sharing file from cards. If the INSERT or MOVE option is used, numeric signs (#) are inserted between the line number and the first character of numeric data.

- OUTPUT - create a card deck from a standard system format, time sharing text file. Numeric signs (#) between the line number and the text are deleted.

The TSCONV program accepts its directive input from file I* and writes its output to file OT. Files I* and OT must be present or an error occurs.

The user-supplied INPUT or OUTPUT directive is printed on the execution report as an indication of which options are being processed.

The INPUT directive begins the card image record and requests the TSCONV program to copy the accompanying card deck onto a specific permanent file. The INPUT directive can begin in any column of the card image and must have no imbedded blanks. The INPUT directive is followed by one or more of the mutually exclusive options listed below:

Option	Result
ASIS,i,j	The text file is generated from the input cards, from the columns specified by _i_ to _j_. Standard columns (default option) for _i_ to _j_ are 1 to 80
MOVE,i,j,m,n	The text file is generated from the input cards, from the columns specified by _i_ to _j_. Line numbers are taken from columns specified by m to n. Standard columns for _i_ to _j_ are 1 to 72, and for _m_ to _n_ are 73 to 80.
INSERT,i,j,m,n	The text file is generated from the input cards and from the columns specified by _i_ to _j_. Lines are sequence numbered, starting with _m_ and incremented by _n_. Standard columns for _i_ to _j_ are 1 to 72. Standard values for both _m_ and _n_ are 10.
ASCII	The text file is generated from input cards, using a binary deck previously punched from this program.
COMDK,option	The text file is generated from input cards consisting of a COMDK (compressed source deck). This option is used in conjunction with the ASIS, MOVE, or INSERT options. If ALTERs are to be made at the time the file is generated, a $ UPDATE card must be employed.
TAB,tab-char,pos-1,pos-2,...pos-n	The TAB specification must appear following the other activity options and separated from the other options by at least one blank. The tab character may be any single character except blank or reverse slant. A reverse slant "\" followed by three digits is interpreted as the octal representation of the ASCII code for the desired tab character. The TAB specification is terminated by the first blank encountered. The tab positions specified must increase in ascending order. TAB supplied with the ASCII option has no meaning, but it is checked for correctness. Any error encountered in tab specification analysis results in a TB abort.

Sample INPUT Control Cards

INPUT,MOVE,1,60,73,80

Text file data is to be taken from columns 1 to 60 of the punched cards and line numbers are to be taken from columns 73 to 80.

INPUT,COMDK,ASIS,1,80

Text file data is to be taken from columns 1 to 80 of the input cards (a COMDK).

The OUTPUT directive is similar in syntax to the INPUT directive in that it must begin the card image record options and may have no imbedded blanks. The OUTPUT directive requests the TSCONV program to produce a card deck from the specified time sharing file. The OUTPUT directive is followed by one or more of the mutually exclusive options listed below:

Option	Result
ASIS,i,j	The text file is read and a BCD card deck is punched in the columns specified by i to j Standard columns (default option) for i to j are 1 to 80.
MOVE,i,j,m,n,1	The text file is read and a BCD card deck is punched, moving data to columns specified by i to j. Line numbers are moved to columns specified by m to n, right-justified. The l specifies the label to be punched starting in column 73, left-justified. Standard columns for i to j are 1 to 72 and for m to n, 73 to 80.
STRIP,i,j	The text file is read and a card deck is punched, stripping off line numbers, with data moved to the columns specified by i to j. Standard columns for i to j are 1 to 80. NOTE: With the above output options, data is converted from ASCII to BCD before punching.
ASCII	The text file is read and a binary deck containing the file text is punched. (See "Binary Card Format" below.)
TAB	See options for INPUT.

Sample OUTPUT Control Card

OUTPUT,ASIS,1,56

The text file is punched into columns 1 to 56 of the card deck.

Definitions

- Each line is punched on a separate card, starting in the column specified (OUTPUT function).
- A line number is an initial string of numeric characters which terminate with a nonnumeric character. Blank is considered a nonnumeric character.
- In the case of the MOVE option, the line numbers are stored right-justified in the columns specified.
- A reverse slash in input processing is treated as a line separator and is replaced with a carriage return.
- The format of a line in a text file is media code 6.

Errors

The following error abort codes are produced by the TSCONV program:

SE ABORT - A binary card is out of sequence. Card number is printed out.

CK ABORT - Checksum of card does not agree with the computed checksum.

NB ABORT - First data card is not binary, but ASCII was specified on control card.

CP ABORT - No control card found (keyword may be misspelled).

TB ABORT - TAB specification error.

DATA LINE TOO LONG FOR I,J FIELD

...portion of the line specified by i to j...

- Occurs on OUTPUT only. If a line of the file is too long for the specified i to j field (i.e., nonblank characters are being discarded), this warning message is issued along with the portion of the line specified by i, j. A maximum of 20 such messages may be given. The complete file is punched, as specified by the i to j field options.

Binary Card Format

Word 1	7/9 punch and number of data words (maximum=21)
Word 2	Checksum
Word 3	Card number, starting at 0
Words 4-24	Text

Sample Deck Setups

The following sample job stream illustrates the copying of an input card deck onto the permanent file FILEOUT. No editing is performed as the entire card images are copied to the output file (ASIS).

```
$  SNUMB   XXXXX
$  IDENT   account number,name
$  USERID  JONES$SECRET
$  PROGRAM TSCONV
$  PRMFL   OT,W,L,JONES/FILEOUT
INPUT,ASIS
      .
(Data deck)
      .
$  ENDJOB
***EOF
```

The following sample job stream accepts an input compressed deck with $ALTER changes and copies the input to output file OUTFIL.

```
$   SNUMB    XXXXX
$   IDENT    account number
$   USERID   JONES$SECRET
$   PROGRAM  TSCONV
$   PRMFL    OT,W,L,JONES/OUTFIL
$   DATA     I*,,COPY
INPUT,COMDK,ASIS,1,80
     .
(Data cards -- COMDK)
     .
$   ENDCOPY
$   UPDATE
     .
(ALTER deck)
     .
$   ENDJOB
***EOF
```

The sample job stream below is representative of a job stream to punch an output card deck from the record images on file TEXTIN.

```
$   SNUMB    XXXXX
$   IDENT    account number, name
$   USERID   JONES$SECRET
$   PROGRAM  TSCONV
$   PRMFL    OT,W,L,JONES/TEXTIN
OUTPUT,ASIS
$   ENDJOB
***EOF
```

TIME SHARING UFAS

A user program can be executed under the Unified File Access System (UFAS) in either batch or time sharing mode. However, not all file formats supported in the batch mode are available in the time sharing mode. The file formats supported in the time sharing mode are the GFRC linked mass storage and the UFF of sequential, relative, indexed, and integrated files. Tape file formats, Indexed Sequential Processor file formats, and the building of UFF indexed alternate keys are not supported in the time sharing mode and no label processing is provided for the file formats supported.

The directory names (symbol references) used to select the UFAS routines to be linked are the same for both the batch mode and time sharing mode.

File Specification

If the file code (fc - two ASCII characters) specified in the file information block macro corresponds to a file already contained in the Available File Table (AFT), the file contained in the AFT will be operated upon. If the file selected is an indexed file, and the file code of the data file is in the AFT, then the file code assigned to the index file in the file information block macro must also be in the AFT.

If the file code is not found in the time sharing Available File Table, by UFAS, the user must supply the operand ASCII descriptor (ADSC9) for the catalog or file string as the file name parameter, FLNAME, in the file information block macro. The file name descriptor, FLNAME, appears in word 9 of the FIB.

The file description must have one of the following formats:

- o Filename
- o Filename$Password
- o USERID/Filename
- o /Catalogname/Filename
- o USERID/Catalogname$Password/.../Filename$Password,Permissions
- o Filename "Alternatename",Permissions
- o Filename,Permissions
- o *Filename
- o *

The input strings for these formats may be in either lowercase ASCII or uppercase ASCII characters. The scan of the file descriptor string is terminated by any non-valid character (see File Management Supervisor manual for valid file name characters), excluding comma, slant, dollar sign, quotation marks, or blanks. The two file strings for an indexed file must be given as: data file string; index file string. The ASCII descriptor in FLNAME must have a total character count, that is, it must include the count of both file strings plus one for the character ";".

File Accessing

File name accessing when the file is not specified in the AFT or is in the AFT, but was not entered by UFAS, is governed by the following rules:

1. If the catalog or file description has a valid alternate name, this name becomes the file name in the AFT.
2. If the catalog or file description has a file name with less than eight characters but no alternate name, the file name is entered in the AFT.
3. Descriptions containing a name with eight characters or less, preceded by an asterisk, are entered in the AFT only if the description occurs as a separate file name without user identification, permissions, or alternate name. The asterisk as a name by itself is converted to *SRC (current file) and entered in the AFT.
4. A file name preceded by a slant (/) cannot be a temporary file.

SECTION IX

BASIC

BASIC (Beginner's All-Purpose Symbolic Instruction Code) is a problem-oriented, algebraic programming language that enables the user to present his program in ordinary mathematical notation, with simple and precise vocabulary and grammar. BASIC is intended to be used with a keyboard-type terminal tied into a time sharing system.

The time sharing system uses a technique by which programs are handled in parallel. A supervisory program acts as a controller of these programs, controlling "stop" and "go" signals to inputs from terminals and preventing demands of one terminal from interfering with demands of other terminals. Thus, time sharing permits a user to work directly with the computer, whether it is within his sight or thousands of miles away.

Time sharing permits a dialogue between the computer and user, permitting the dialogue to begin immediately, without waiting for the computer to complete previous programs. Data is fed from the terminal directly to the computer and answers are received quickly at the same terminal.

If the program contains a mistake, the computer informs the user.

The program can be corrected or changed by the user as if conversation was taking place by phone, except in this case, the conversation is typed or displayed, depending upon the type of terminal in use.

Because BASIC is such a simple programming language and because time sharing permits the correction and completion of most problems within minutes, BASIC as used in a time sharing system provides a highly satisfactory computation environment for both the novice and experienced programmer.

STATEMENT DEFINITION

Each BASIC statement consists of the following elements arranged in the order given:

Statement (or Line) number - by its ascending order, indicates the processing sequence of the statement.

BASIC word - specifies the computer operation to be performed.

Parameters - in most statements are variables, expressions, and numbers used in, or to direct the operation performed by, the statement.

MATHEMATICAL NOTATION AND OPERATIONS WITHIN A STATEMENT

Variable Representation

In the BASIC language, a variable can be represented by

1. a letter
2. a letter and a digit
3. either of the above, followed by the character $

For example A,Z,K6, and X may represent variables, but AR, Z12, 6K, and 22 cannot. The inadvertent use of the digit 0 for the letter O (and vice versa) in a variable causes errors in a program; use of the letter O or the digit 0 in variable representation is not recommended. The user may find the choice of a letter as a mnemonic for a variable helpful; for example, P for price, S for sales, and N for numbers.

Variables with $s are restricted to the assignment of strings (alphanumeric data) and are referred to as "string variables," in contrast to variables without the $ that are referred to as "numeric variables." Numeric variables, when used as a starting point in calculations (e.g., for a counter), have an initial value of zero. String variables have an initial value of zero when used for character count.

A BASIC variable is assigned a value, during the execution of a program, from the numbers given in a related LET, FOR, READ, or INPUT statement. It retains this value during the processing, unless it is reassigned a new value by another of these statements.

List And Table Variables

Subscripted variables are represented in BASIC as

variable name (subscript)
or
variable name (subscript, subscript)

where the subscript can be an integer, variable, or an arithmetic expression such as (1+K) or (A(3,7),B-C). The subscript must always be enclosed by parentheses. Subscript values should begin at 1 (i.e., not 0).

A list variable designates an element of a one-dimensional array that can be represented by such as P(15), P(H) or L(20). Before a list variable can be used in any statement, the maximum value of its subscript (i.e., size of list) must be specified in a DIM statement; otherwise a list of 10 or less is implied.

A table variable designates an element of a two-dimensional array that can be represented by such as S(15,17) or T(20,30). Before a table variable can be referenced in any statement, the maximum value of its subscripts must be specified in a DIM statement; otherwise, subscripts of 10 or less are implied.

Specification of the values of subscripts for list variables or table variables in DIM statements is not required if subscripts of 10 or less occur. BASIC provides for automatic dimensioning in such cases. Automatic dimensioning assigns a value of 10 for the subscript of the list variable and a value of 10 by 10 for the array of a table value. If a subscript with a value greater than 10 is used with a list or table variable and the list or table variable is not dimensioned in a DIM statement, an error message is generated. Conversely, if values of subscripts less than 10 are specified in DIM statements, no adverse programming effects result.

Use Of Numbers

A number can be positive or negative, can contain up to nine digits, and must be in decimal form. BASIC accepts 0.01, 2, -3.675, 123456789, -.987654321, and 483.4156 as numbers, but rejects 14/3 (this is an expression) or 32,437 (as representing 32437). Numbers are stored as single-precision floating-point values. Thus, the maximum value that can be represented accurately is 134217727; larger values are only approximated since digits beyond the eighth position are not reliable.

A number can also be expressed in "E notation," equivalent to expressing it as a power of 10. For example, in E notation,

0.00123456789	can be	0.123456789E-2 or 12.3456789E-4
1967	expressed	1.967E3 or 19.67E2
10,000,000	as	1E7 or 100E5

The decimal point can be positioned anywhere within the number as long as the integer following the E indicates its correct position. Note that E and an exponent alone cannot represent a number. For example, E7 cannot be written as a number to represent 10,000,000; it must be written as 1E7 to indicate 1 multiplied by 10 to the 7th power.

Arithmetic Operations

Five arithmetic operations can be performed by BASIC. Each of the following symbols represents an arithmetic operation that can be included in an expression.

Operator symbol	denotes	as illustrated by
+	addition	A + B
-	subtraction	A - B
*	multiplication	A * B
/	division	A / B
** or ↑	raise to a power	A**B or A ↑ B

Relational Symbols

Six relational tests can be made with BASIC. Symbols representing these relationships can be used in statements when comparisons are required. The symbols and illustration of their use follow.

Relational symbol	denotes	as illustrated by
=	is equal to	A = B
<	is less than	A < B
<= or =<	is less than or equal to	A <= B or A = <B
>	is greater than	A > B
>= or =>	is greater than or equal to	A>= B or A = >B
<>or<>	is not equal to	A< >B or A< >B

Those terminals that lack the symbols for less than or greater than characters can make use of an alphabetic code to obtain required relational symbols.

Relational Code	Denotes	As Illustrated By
EQ	is equal to	A EQ B
LT	is less than	A LT B
LE	is less than or equal to	A LE B
GT	is greater than	A GT B
GE	is greater than or equal to	A GE B
NE	is not equal to	A NE B

Use Of Expressions

The computer performs its primary function (that of computation) by evaluating expressions contained within program statements. These expressions are similar to those used in standard mathematical notation with the exception that all BASIC expressions must be complete within a statement and a statement is restricted to a single line. Expressions are made up of numbers, variables, operations, and functions by themselves or in conjunction with one another.

The user must understand the order in which the computer does its work. For example, if the input is A + B * C ** D, the computer first raises C to the power D, multiplies this result by B and then adds A to the resulting product. This is the same convention as is usual for A + B times C raised to the power D. If this is not the order intended, then parentheses must be used to indicate a different order. For example, if the product of B and C raised to the power D is required, the user writes A + (B * C) ** D; or, if one wants to multiply A + B by C to the power D, the user writes (A+B)C**D. The user could even add A to B, multiply their sum by C, and raise the product to the power D by writing ((A+B) * C) ** D. The order of arithmetic priorities is summarized in the following rules.

1. The expression inside parentheses is computed before the parenthesized quantity is used in further computations.

2. In the absence of parentheses in an expression involving addition, multiplication, and the raising of a number to a power, the computer first raises the number to the power, then performs the multiplication, and the addition comes last. Division has the same priority as multiplication, and subtraction the same as addition.

3. In the absence of parentheses in an expression involving only multiplication and division, the operations are performed from left to right, as they are read. The computer also performs addition and subtraction from left to right.

In practice, extensive use of parentheses tend to eliminate most ambiguities that may arise.

Mathematical Functions

BASIC provides for standard mathematical functions. Each is represented by a three-letter mnemonic of its name and is followed by an expression enclosed in parentheses. The user need only enter the function in a statement to obtain its computed value in a run of a program.

Function	means find the
SIN(X)	sine of X
COS(X)	cosine of X
TAN(X)	tangent of X
COT(X)	cotangent of X
ATN(X)	arctangent of X
EXP(X)	e to the power X
LOG(X)	natural logarithm of X
CLG(X)	common logarithm of X
ABS(X)	absolute value of X
SQR(X)	square root of X

In these definitions, the letter X represents an expression, which, for the trigonometric functions, implies an angle measured in radians. If the value of X in LOG(X), CLG(X), or SQR(X) is negative, then the negative sign is ignored, the positive value is used, and an error message is printed.

Four additional mathematical functions are included in BASIC.

Function	means
INT(X)	truncate X
RND(X)	produce a random number
SGN(X)	sign determination
DET(X)	provide determinant of last matrix inverted

In addition, the user can employ the DEF statement to define one or more of his own functions.

Miscellaneous Functions

A set of miscellaneous functions is available for use to provide a variety of non-mathematical operations. These are as follows:

Function	means obtain
TIM(X)	elapsed processor time
CLK$	time of day
DAT$	calendar date
NUM(X)	count of matrix data elements
SST(X$,Y,Z)	selected characters of a string (substring)
TAB(X)	character print position
SPC(X)	space print position
LEN(X$)	number of characters in string
LIN(X)	last line number encountered in reading/writing file
ASC(X)	numeric value of character or abbreviation
STR$(N)	expression to string conversion
VAL(S$)	string to expression conversion
TST(S$)	nonzero output if string can be interpreted as a number
HPS(X)	horizontal point position of next field, in current line, f file being written

STATEMENT DESCRIPTIONS

Purpose: A concise statement of the operation it performs.

Format: The general form for its use in the program, with the literal entries in CAPITAL letters and descriptive names for variable entries in lowercase letters enclosed within the symbols . Parentheses are to be inserted as indicated. Note that an expression can be either a simple variable or a formula.

Examples: Typical uses are given to explain and clarify the format. Statement numbers are arbitrary and are used for illustrative purposes.

Rules: Requirements and cautions concerning the use of the statement.

Remarks: Pertinent comments related to the uses of the statement.

Arithmetic Statements

DEF

Purpose: To define a function that is to be used repeatedly within a given program.

Format: DEF FN_ (variable) = <expression>

Example: *10 DEF FNG(Z) = 1 + SQR(1+Z*V)

Rules: 1. The variable must be unsubscripted.

2. Up to 26 functions can be defined within a single program; i.e., FNA, FNB, ..., FNZ.

3. The space following FN is to be filled with any alpha character.

Remarks: If a function requires more than one line for its definition, a multiple-line defined function can be written.

LET

Purpose: To evaluate an expression and assign the resultant value to a specified variable.

Format: LET <variable> = <expression>

Examples:

1. `*10 LET X=X+1`

2. `*20 LET W7=(W-X4+3)*(Z-A)/(A-B)-17`

3. `*30 LET X(6)=0`

Remarks: The LET statement is not a statement of algebraic equality; it is an assignment or replacement statement.

A variable defined in a LET statement can be subscripted or unsubscripted.

Multiple variable replacement is permitted within a LET statement. For example:

```
*10 LET A=B=C
*20 LET A=B=C=100
*30 LET A(I)=B(X+Y/Z)=C(J)
*40 LET A(B(J))=B(J)=C(5)
*50 LET E$=F$=G$
*60 LET E$=F$=G$="MULTIPLE REPLACEMENT"
*70 LET H$(B(J))=H1$="EXAMPLES"
```

Replacement is executed on a right-to-left basis. A numeric BASIC variable cannot be replaced by a string variable and vice versa. Multiple replacement is limited to 20 elements within one LET statement.

The BASIC word LET can be implied; i.e., the statement

```
*10 X=X+1
```

implies LET precedes the variable X and is a valid assignment statement.

MAT

Purpose: To request the system to compute or manipulate a matrix.

Format: MAT READ <variable or comma-separated variables>

MAT PRINT <variable or comma-separated variables>

MAT INPUT <variable>

MAT <variable> = operation

Remark: A detailed description of the use of MAT statements in operations upon matrices is given under "Matrices" later in this section.

Specification Statements

CHANGE

Purpose: To permit translation of data from numeric code representation to its equivalent string character and, conversely, string character to numeric code.

Format: CHANGE <variable> TO <variable>

Examples:

1. `*10 CHANGE A TO A1$`

   Elements of numeric variable A are converted to characters and stored in string A1$.

2. `*20 CHANGE Z5$ TO X`

   Characters in string Z5$ are converted to their numeric equivalents and stored in the elements of X.

Rules:

1. One variable must be a numeric variable, the other a string variable.
2. The number of characters to be converted is limited to 132.
3. If a numeric variable has not been previously dimensioned, it is automatically dimensioned by 10.
4. When the conversion is to be from a numeric code list to a character string, the user must provide a count of the number of elements to be converted. This is done prior to the CHANGE command by an assignment statement that stores the desired count in element (0) of the numeric array.

   For example:

   ```
   *10 LET A(0) = 15
   *20 CHANGE A TO A1$
   ```

   directs the program to convert 15 of the numeric elements in list A to their related characters and concatenate them in string A1$.

If the count specified for conversion is smaller than the number of items in the numeric list, the remaining characters are truncated; if the count given is larger, the string contains irrelevant information.

5. When a string is converted to numerics, a count is not specified. The complete string is converted if the numeric array is of sufficient length. If the array dimension is smaller than the string length, an error message occurs at execution time. If the string characters do not fill the entire array, the remaining array elements remain unchanged.

6. A table of characters and equivalent codes can be found under "Alphanumeric Data and String Manipulation" later in this section.

DATA

Purpose: To specify numeric values for variables in a READ statement.

Format: DATA <number or comma-separated numbers>

Example:

```
*10 READ A,B,X,L1,Z
     . .
     . .
*100 DATA 1,3.4,7,-167.921,1.9E5
```

Rules:

1. Only numbers (positive or negative) are allowed; numbers can be written conventionally or with E-notation.

2. The numbers in the DATA statement must be in the same sequence as the respective variables in the associated READ statement (in the example, X = 7).

3. The numbers can be in one or more DATA statements, but the sequence must correspond to that for the variables in the READ statement. That is, the DATA statement in the example could be replaced by as many as five DATA statements.

Remarks: DATA and READ statements are always used jointly.

The collection of all numbers in all of the DATA statements of a program is referred to as a "data block."

The placement of DATA statements in a program is arbitrary; common practice is to collect all of the DATA statements in one place in the program.

DIM

Purpose: To define the dimension(s) of a list or table and thereby reserve sufficient space in the computer.

Format:

1. For a list

   DIM <variable> (subscript)

2. For a table

   DIM <variable> (subscript, subscript)

Examples:

1. *10 DIM H(35)

   This statement reserves 35 computer locations.

2. *20 DIM Q(5,25)

   This statement reserves 125 computer locations, since it involves 5 items times 25 items, as in 5 x 25 table.

Space for more than one list and/or table may be defined in a single DIM statement.

```
*30 DIM M(50), R(25,35), T(10,10)
```

Rules:

1. A subscripted variable must appear in a DIM statement to achieve explicit dimensioning; otherwise, automatic dimensioning (subscript value of 10 or less) is implied.
2. DIM statements defining variables must precede the use of these variables.
3. The dimension(s) of a list or table in a DIM statement must be expressed explicitly; expressions are not to be used as subscripts.
4. For a list, the variable can be numeric or string; for a table, the variable must be numeric.

Input/Output Statements

INPUT

Purpose: To permit the input of desired values of variables during program execution time.

Format: INPUT <variable or comma-separated variables>

When, in the execution of the program, this statement is reached, a question mark is printed. The user must then enter a number or sequence of numbers before the program can continue.

Example: *10 INPUT X,Y,Z is entered into the program as a statement

? but only a question mark appears during execution; the user must then type the comma-separated values of X, Y, and Z after the question mark.

Rules:

1. Each INPUT statement must be positioned logically ahead (in the order of processing) of the statement that is to use the data values requested.

2. The numbers listed after the question mark must also be separated by commas.

3. The numbers must be typed in the same sequence as the variables to which they are assigned.

PRINT

Purpose: To instruct the system to perform one of the following print operations:

1. Print out the result of computations.

2. Print out text, verbatim, to supply such items as messages, information, or labels.

3. Print out a combination of uses 1 and 2.

4. Skip a line in the printout of program execution.

Format: Every PRINT statement begins with the BASIC word PRINT but can vary in form, depending upon the print operation required.

Example 1:

```
*10 PRINT X,SQR(X)
```

results in the printing of the value of X, and a few spaces to the right of that number, its square root.

```
*20 PRINT B*C,EXP(A),Y/Z,E+F,X**2
```

results in the printout of 5 computed values.

Example 2:

Whenever text is to be printed verbatim during the execution of a program, it is enclosed within quotation marks in the statement; whatever is enclosed is reproduced, including spaces and punctuation. This verbatim text is referred to as a label.

```
*40 PRINT "NO UNIQUE SOLUTION"
```

results in the printout

```
NO UNIQUE SOLUTION
```

Example 3:

```
*50 PRINT "THE VALUE OF X IS", X
```

results in the printout, if X = 3,

```
THE VALUE OF X IS    3
```

```
*60 PRINT "THE SQUARE ROOT OF" X, "IS" SQR(X)
```

results in the printout, if X = 625,

```
THE SQUARE ROOT OF 625 IS 25
```

Example 4:

When a statement such as

```
*70 PRINT
```

is encountered by the program during its execution, the terminal carriage is advanced one line at that stage of program execution.

Remarks: The form in which BASIC prints numbers is not under the control of the user. The following items apply to the printing of numbers when PRINT statements are utilized.

1. When a number is an integer, the decimal point is not printed.

2. When a computed value consists of an integer with more than seven digits, BASIC prints

   o the first significant digit

   o followed by a decimal point

   o the next five digits (the integer is rounded)

   o the letter E

   o followed by a space

   o and finally, a number indicating the power of 10 (how many places the decimal point is to be moved to the right).

   For example, the integer

   32437528259 becomes 3.24375E 10 when printed.

3. No more than seven significant digits are printed.

4. Numbers less than 1.0 are printed with a decimal point followed by up to seven significant digits.

   For example,

   .1234567

   would be printed exactly as shown, whereas the number

   .01234576978

   would be rounded and printed as

   .0123458

5. Numbers less than 0.0001 are printed in E-format.

   For example,

   .00001234567

   would be rounded and printed as

   1.23457E-05

The PRINT statement can be modified by the use of:

- commas
- semicolons
- function TAB(X)
- function SPC(X)

in order to vary the format of the output.

PRINT USING

Purpose: To instruct the system to print out a formatted line.

Format: PRINT USING <statement number, output list>

Where:

"statement number" is number of a statement in the program that contains format control characters and printable constants; "output list" consists of comma-separated arguments to be printed in sequential order.

Example:

```
*10 A = 100
*20 B = 200
*30 C = -300
*40 D$ = "END OF LIST"
*50 PRINT USING 60,A,B,C,D$
*60:    'LLLLLLLLLLLLLL
*70 END
*RUN

ƀƀ100ƀƀ200ƀ-300ƀEND OF LIST
```

Rules:

1. The statement number named in a PRINT USING statement points to an "image" statement that formats the line to be printed. The image statement is of the form

   statement number: image

2. The image of an image statement (colon-separated from the statement number) consists of format control characters and printable constants.

3. Format control characters are as follows:

   ' (apostrophe) - a 1-character field that is filled with the first character in an alphanumeric string, regardless of string length.

   # (number sign) - the replacement field for a numeric character; each # specifies a space for one digit; a # specifies space for the minus sign if sign is present.

   (four up-arrows) - specifies scientific notation for a numeric field (E-format).

4. Printable constants are all characters other than format control characters.

Remarks: The image of an image statement can consist of one or more of the following fields:

integer
decimal
exponential
alphanumeric
literal

READ

Purpose: To read values listed in DATA statements and assign them to specified variables.

Format: READ variable or comma-separated variables

Example:

```
*10 READ A,B,X,L1,Z
  .  .
  .  .
  .  .
*100 DATA 1,2,7,2,-167.921
```

Rules:

1. Each READ statement must be positioned logically ahead (in the order of processing) of the arithmetic or PRINT statement that is to use the data requested.
2. The variables in a READ statement must be in the same sequence as the respective values in the associated DATA statement (in the example, 7 is assigned to X).

Remarks: READ and DATA statements are always used jointly. If there are not enough numbers in the data block (collection of DATA statements) for the variables in a READ statement, then the program is assumed to be finished, no further processing of data occurs, message OUT OF DATA is printed, and the program terminates processing.

If a READ statement is executed more than once, as if in a loop, the data block supplies the next available number for each execution, unless a RESTORE statement is executed.

RESTORE

Purpose: To restore the data block to its original state, so that it can be read by a logically subsequent READ statement and thus used for further processing.

Format: RESTORE

Example: In the following portion of a program

```
*100 READ N
*110 FOR I = 1 TO N
*120 READ X
  .  .
  .  .
*200 NEXT I
  .  .
  .  .
*560 RESTORE
*570 READ X
*580 FOR I = 1 TO N
*590 READ X
  .  .
  .  .
*650 DATA 4, 15, 35, 23, 9
*660 END
```

the data is read, the data block is then restored to its original state, and the data is then read again for processing. Statement 570 is used to pass over the value of N, since it is already known.

Remarks: When the program is executed, the data from the DATA statements are saved in memory as a data block. The data is then assigned to variables via a READ statement in the sequence given. The RESTORE statement directs the computer to reassign data starting from the beginning of the data block; if this statement were not present in the above example, then the system would stop processing at statement 570 and print out the message OUT OF DATA.

CALL

Purpose: To call a program, previously saved on a permanent file, for use as a subroutine within the primary program.

Format: CALL <filename, password>

Example:
```
*10 DEF FNP(X,Y)=SQR(X*X+Y*Y)
*20 CALL SUB1
*30 DATA 3
*40 END
```

Program SUB1, previously saved, is as follows:

```
*10 READ B,C
*20 IF B=0 THEN 70
*25 CALL SUB2
*30 LET A=FNP(B, )
*40 PRINT "HYPOTENUS  ";A
*50 GOTO 10
*60 DATA 4,0,0
*70 RETURN
```

Program SUB2, previously saved, is as follows:

```
*10 IF B   0 THEN 40
*20 PRINT "NEGATIVE ARGUMENT"
*30 STOP
*40 IF C   0 THEN 20
*50 RETURN
*60 END
```

Rules:

1. All variables and functions must be common to the primary (calling) program and the called programs.

2. The return from a called program to the calling program must be by the way of a RETURN statement.

Remarks: A password is required only if one is attached to the filename.

Multiple returns are permitted within a called program. The return is always to the statement immediately following the CALL statement. A called program can call other programs.

An END or STOP statement to terminate execution can be in either the calling or called program.

Line numbers in calling or called programs are completely independent.

DATA statements are compiled from the primary program first, and then from each of the called programs in the order in which the CALL statements are encountered.

A total of 15 different programs can be called from the primary and called programs.

FOR and NEXT

Purpose: The FOR statement is the initial statement of a program loop and it specifies the variable used to count the iterations through the loop, its range of values, and the stepsize for each pass through the loop. The NEXT statement is the last statement in the loop and it directs the processing to either repeat the loop or continue sequential execution if the specified number of iterations have been completed.

Format:

```
FOR <variable> = <expression> TO <expression>
STEP <size expression>
  .   .
  .   .
  .   .
NEXT <variable>
```

<varible> specifies an unsubscripted loop-control variable. <expression> TO <expression> specifies the range of values to be assigned to the variable. The first expression sets the initial value of the variable; the second expression sets the final value of the variable. For a positive stepsize, the loop is repeated until the variable reaches a value greater than or equal to the final value. For a negative stepsize, the loop is repeated until the variable reaches a value less than or equal to the final value. STEP <size expression> specifies the increment or decrement to be added to the loop-control variable on each pass through the loop; if STEP and its size expression are omitted, the increment is assumed to be 1.

Examples:

1.
```
*30 FOR X = 1 TO 25
  .  .
  .  .
*80 NEXT X
```

2.
```
*120 FOR X4 = (17+COS(Z)/3) TO 3*SQR(10) STEP N*Z
  .  .
  .  .
*235 NEXT X4
```

3.
```
*240 FOR Z = 8 TO 3 STEP -1
  .  .
  .  .
*300 NEXT Z
```

4.
```
*450 FOR J = -3 TO 12 STEP 2
  .  .
  .  .
*500 NEXT J
```

5.
```
*30 FOR X = 0 TO 25 STEP A
  .  .
  .  .
*80 NEXT X
```

Rules:

1. If the range requires a negative step and it is omitted, the body of the loop is executed once for the initial value of the variable. The variable is tested after the first time the implied step (+1) is added, and is found to exceed the termination condition.

2. Paired FOR and NEXT statements must specify the same loop-control variable.

GOSUB and RETURN

Purpose: GOSUB - To direct the system to the first statement of a subroutine sequence that is located elsewhere in the program (i.e., to "call" a subroutine).

RETURN - To return the processing to the next statement following the GOSUB statement used to call the subroutine.

Format: GOSUB number of first statement of subroutine

Example:

```
*80 GOSUB 200
*90 LET X = 5
   . .
   . .
*200 LET X = INT(A/B)
   . .
   . .
*350 RETURN
```

Statement 350 returns the processing to statement 90.

Remarks: A subroutine can be placed anywhere within a program but should only be entered by the way of a GOSUB statement. Return from a subroutine must be by the way of a RETURN statement; no other type of statement can be used.

Logic Statements

GOTO

Purpose: To transfer unconditionally to a statement other than the next one in the processing sequence.

Format: GOTO <statement number>

Example: *50 GOTO 20

Remark: The GOTO statement can be used as a means of delegating a program to return repeatedly to blocks of instructions.

IF-----THEN
or
IF-----GOTO

Purpose: To direct the system to either go to a designated out-of-sequence statement if a certain condition is met or proceeds to process in sequence, thus providing a 2-way conditional switch.

Format: IF <expression> relation <expression> $\left\{\begin{array}{l}\text{THEN}\\ \text{GOTO}\end{array}\right\}$ <statement number>

Examples:

1. *10 IF SIN(X) = M THEN 80 or
   *10 IF SIN(X) = M GOTO 80

2. *20 IF G=0 THEN 65 or
   *20 IF G=0 GOTO 65

In each example, if the condition is met, then the computer transfers to the designated statement number; otherwise, it proceeds to process the next statement in sequence.

Rule: BASIC provides six relational tests. The following symbols representing relationship can be used in IF----THEN or IF----GOTO statements when comparisons are required.

Relational Symbol	Denotes	As Illustrated By
=	is equal to	A = B
<	is less than	A < B
<= or =<	is less than or equal to	A< = B or A = <B
>	is greater than	A > B
>= or =>	is greater than or equal to	A> = B or A = >B
>< or ><	is not equal to	A >< B or A <> B

Those terminals that lack the less-than or greater-than characters can make use of an alphabetic code to obtain required relational symbols.

Relational Code	Denotes	As Illustrated By
EQ	is equal to	A EQ B
LT	is less than	A LT B
LE	is less than or equal to	A LE B
GT	is greater than	A GT B
GE	is greater than or equal to	A GE B
NE	is not equal to	A NE B

ON-----THEN
or
ON-----GOTO

Purpose: To direct the system to go to designated statements, thus providing a multiple switch.

Format: ON <expression> {THEN | GOTO} <statement numbers>

Examples:

1.
```
*10 ON X GOTO 100,200,150
```

if X=1, the system branches to statement 100
if X=2, to statement 200
if X=3, to statement 150

The value of X is dependent upon conditions set in another part of the program.

2.
```
*110 FOR X = 1 TO 3
*120 ON X GOTO 200,300,400
*200 PRINT "A"
*210 GOTO 500
*300 PRINT "B"
*310 GOTO 500
*400 PRINT "C"
*500 NEXT X
*600 STOP
*900 END
*RUN
    A
    B
    C
```

Rules:

1. Any number of statement numbers can follow THEN or GOTO, providing they fit on one line.
2. Statement numbers following THEN or GOTO can be repeated.

Remarks: The expression can be a variable or a formula. The variable must be an integer ranging from one to the number of statement numbers specified. For a formula, computation is made and its integer part is taken as the value. If the integer part is less than one or is larger than the number of statement numbers specified, an error message is printed.

STOP

Purpose: To stop the execution of the program.

Format: STOP

Example: *250 STOP

. .

*340 STOP

. .

*990 END

This example illustrates that there can be more than one STOP statement within a program, and if any one is processed, the program is terminated.

Remark: STOP is the equivalent of GOTO XXXX, where XXXX is the line number of the END statement in the program.

END

Purpose: To indicate the end of a program.

Format: END

Example: *990 END

Rules:

1. The END statement is optional in a program.
2. The END statement, if used, must have the highest line number of the program.
3. The END statement, if omitted, is simulated when the RUN command is given and an end-of-file situation is detected.

Remarks: In the execution of the program, the system recognizes the END statement as a command to terminate output. The END statement can be reached during program execution by normal sequential processing, or by program control being transferred to it by means of a GOTO or STOP statement.

Utility Statements

CHAIN

Purpose: To permit sequential compilation and execution of a series of BASIC programs.

Format: CHAIN <filename, password, line number>

Examples:

1. *10 CHAIN FILE1,PASS1,100
2. *20 CHAIN A$,PASS2
3. *30 CHAIN B$,1234,

Rules:

1. The filename can be expressed in the following manner:

   a. in ASCII characters, a limit of eight characters

   b. enclosed in quotes; i.e., "<u>filename</u>"

   c. as an alphanumeric variable, subscripted or unsubscripted, with the values of the variable and subscript (if any) assigned at compilation or execution times.

2. If a file with a password is named in a CHAIN statement, the password must accompany the filename.

3. The CHAIN statement permits chaining to a line number within a file.

4. Each CHAIN statement is restricted to one filename.

5. If a password is all numeric and no line number is specified, the password must be delimited by a trailing comma; otherwise, the password is interpreted as a line number.

Remarks: The current file and a file named in a CHAIN statement must be files saved prior to any attempt to perform the chaining function.

If a line number is given in a CHAIN statement, it must be given as a numeric value.

There is no limit to the number of programs the user desires to compile and execute by means of CHAIN statements.

The use of double quotes to enclose a filename permits compatibility with programs written for other systems.

TRACE ON

TRACE OFF

Purpose: To instruct the system to print out the line numbers, at execution time, of those statements enclosed between a TRACE ON and TRACE OFF statement.

Format:

```
TRACE ON
  .
  .        sequence of statements
  .
TRACE OFF
```

Example:

```
*10 LET X=0
*20 IF X    0 GOTO 80
*30 TRACE ON
*40 LET X=15
*50 PRINT "PHASE 1"
*60 GOTO 20
*70 TRACE OFF
*80 PRINT "PHASE 2"
*90 END
```

When RUN is given as a command, program execution will be as follows:

```
* AT 40
* AT 50
PHASE 1
* AT 60
PHASE 2
```

Remarks: A TRACE ON statement can be used without a TRACE OFF statement; i.e., the END statement simulates a TRACE OFF statement. If a TRACE OFF statement is encountered before a corresponding TRACE ON statement, that TRACE OFF statement is ignored.

Multiple TRACE ON-TRACE OFF statements can be made within one program.

Documentation Statement

REM

Purpose: To permit the insertion of an explanatory remark in a program.

Format: REM <followed by the remark>

Example:

```
*50 REM  INSERT DATA IN LINES 900-1000.
*60 REM  THE FIRST NUMBER IS N, THE
*70 REM  NUMBER OF POINTS REQUIRED.
```

Remarks: The computer stores the text of the REM statement and does not process it. A GOSUB, IF-----THEN, or GOTO statement can refer to a REM statement by referencing its statement number. When a remark exceeds a line, a statement number and REM must be typed on each succeeding line before continuing the remark.

Programs containing distinctive parts such as subroutines or loops should have these parts labeled by means of REM statements. Such labeling readily identifies sections of a lengthy program and permits the user to rapidly scan the program if corrections or additions are required.

A BASIC PROGRAM EXAMPLE

The first step in writing a BASIC program is to analyze the problem and determine the exact operations that must be performed to produce the desired results. Having determined the required operations, it is then necessary to convert them into BASIC statements.

This example describes the preparation of a BASIC program that calculates and prints out the average number of miles traveled by a vehicle per gallon of gasoline, given:

Old Miles	New Miles	Gallons of Gasoline Used	Average Number of Miles per Gallon
3332	3553	14.8	?
	3801	7.4	?
	3926	15.2	?
	4091	11.3	?
	4275	10.9	?
	4460	9.8	?
	4628	9.8	?
	4864	12.3	?
	5250	13.6	?
	5617	6.7	?
	6112	10.0	?
	6379	14.0	?

Overall average miles traveled per gallon of gasoline ?

Analyzing The Problem

An analysis of the problem indicates that the following operations should be performed to arrive at the solution:

1. Show five column headings across the typeout as follows:

   Old Miles
   New Miles
   Miles Traveled
   Gallons of Gasoline Used
   Average Miles Traveled per Gallon of Gasoline

2. Write given "old miles" value in column 1.

3. Write first given "new miles" value in column 2.

4. Write first given "gallons of gasoline" value in column 4.

5. Subtract value in column 1 from the value in column 2 and write the result in column 3.

6. Divide value in column 3 by value in column 4 and write the result in column 5. This is average number of miles traveled per gallon of gasoline.

7. Move down to second line in each column.

8. Write first given "new miles" value in column 1.

9. Write second given "new miles" value in column 2.

10. Write second given "gallons of gasoline" value in column 4.

11. Subtract last value in column 1 from last value in column 2 and write result in column 3.

12. Divide last value in column 3 by last value in column 4 and write result in column 5.

13. Move down to third line in each column.

.
.
.

Continue writing of appropriate values in proper columns and making computations until all data is utilized. Move down to next line after completing each "average miles traveled per gallon of gasoline" computation and writing of result in column five.

.
.
.

14. Divide total number of miles traveled by total gallons of gasoline used and title the result "Overall average miles traveled per gallon of gasoline."

Converting To BASIC Language

Having determined the required operations, it is now necessary to convert the operations into BASIC statements.

The following relationships and abbreviations will facilitate the writing of the program:

$M = N-L$ and $A = \frac{M}{G}$ where:

M = miles traveled

L = old miles

N = new miles

A = average miles per gallon

G = gallons of gasoline

The following sequence of statements can now be written.

```
5   REM TOTAL MILES/GALS
10  PRINT"OLD MILES ";"NEW MILES ";"MITR ";"GAL GAS ";"AMPG"
20  PRINT"-----------------------------------------------------"
30  READ L
40  LET L1 = L
50  READ N
60  IF N=0 THEN 190
70  READ G
80  LET M=N-L
90  IF M=0 THEN 120
100 LET A=M/G
110 IF A >< 0 THEN 130
120 PRINT "YOUR TANK HAS A HOLE IN IT"
130 IF A < 35 THEN 150
140 PRINT "I DONT BELIEVE IT"
150 PRINT L;N;M;G;A
160 LET L=N
170 LET G1=G1+G
180 GOTO 50
190 PRINT "TOTAL MILES/GALS",(L-L1)/G1
200 DATA 3332,3553,14.8,3801,7.4,3926,15.2,4091,11.3,4275
210 DATA 10.9,4460,9.8,4628,9.8,4864,12.3,5250,13.6,5617
220 DATA 6.7,6112,10.0,6379,14.0,0
230 END
```

Explanation Of The Statements

5 REM TOTAL MILES/GALS

Identifies the program; does not enter into the execution process.

10 PRINT "OLD MILES ";"NEW MILES ";"MITR ";"GAL GAS ";"AMPG"

20 PRINT "---"

Statements 10 and 20 direct the system to print verbatim that information enclosed by quotation marks.

30 READ L

Assigns the first value in the data block to variable L; i.e., 3332 to L (old mileage).

40 LET L1=L

Assigns the existing value of L which is 3332, to L1. The value assigned to L changes as the program execution progresses but the value assigned to L1 will remain 3332. It is necessary to preserve the 3332 value for calculating total miles traveled; statement 190 directs the computer to make this computation.

50 READ N

Assigns the next value in the data block to variable N; i.e., 3553 to N (new mileage).

60 IF N=0 THEN 190

Directs the system to execute statement 190 instead of statement 70 if the value assigned to N in statement 50 was 0; i.e., last entry in data block.

70 READ G

Assigns the next value in the data block to variable G; i.e., 14.8 to G (gallons of gasoline)

80 LET M=N-L

Directs the system to subtract the value of L from the value of N and assign the difference to variable M (miles traveled).

90 IF M=0 THEN 120

Directs the system to execute statement 120 instead of statement 100 if the value assigned to M in statement 80 was 0.

100 LET A=M/G

Directs the system to divide the value of M by the value of G and assign the resulting value to A (average miles per gallon).

110 IF A >< 0 THEN 130

Directs the system to execute statement 130 next instead of statement 120 if the value assigned to A in statement 100 was any value other than 0.

120 PRINT "YOUR TANK HAS A HOLE IN IT"

Directs the system to print out, verbatim, that information enclosed by quotation marks. This statement is executed only if the value assigned to A in statement 100 was 0, or if the value assigned to M in statement 90 was 0.

130 IF A < 35 THEN 150

Directs the system to execute statement 150 instead of statement 140 if the value assigned to A in statement 100 was less than 35.

140 PRINT "I DONT BELIEVE IT"

Directs the system to print out, verbatim, information enclosed by quotation marks. This statement is executed only if the value assigned to A in statement 100 was equal to or greater than 35.

150 PRINT L, N, M, G, A

Directs the system to print, in column form, the values of L, N, M, G, and A assigned in statements 30, 50, 80, 70, and 100, respectively.

160 LET L=N

Assigns the existing value of N (new mileage) to L (old mileage) in preparation for the next calculation.

170 LET G1=G1+G

Establishes a means for recording the accumulative gallons of gasoline used for the entire trip. As there was no READ statement to assign a value, the computer initially set G1 to zero.

On the first pass through the data block, G was assigned the value 14.8. This statement directs the computer to add the value of G (14.8 in this instance, assigned in statement 70) to the initial value of G1 (zero), establishing a new value for G1 (14.8). On the second pass through the data block the next value of G (7.4) is added to the existing value of G1 (14.8) establishing another new value for G1 of 22.2. This summation of G and G1 is repeated on subsequent passes as long as there are new values of G in the data block.

180 GOTO 50

Directs the system to go to line 50, thus repeating the same sequence of statements over again to find the average miles traveled per gallon of gasoline for the next refueling. Eventually, a value of N equal to zero is achieved and statement 60 is executed. At that point, control of the program is given to statement 190.

190 PRINT "TOTAL MILES/GALS", (L-L1)/G1

Instructs the system to calculate and print the overall miles traveled per gallon of gasoline for the entire trip.

The statement accomplishes this by directing the system to subtract L1 (3332 from statement 40) from L (6379 - the last old mileage assignment in the data block) and then divide the difference by G1 (accumulative gallons of gasoline calculated in statement 170).

200, 210, 220 DATA

Data statements are not executed. They are used to enter the data required for the subsequent execution of the program. The arrangement in which the data is entered in the statement is critical because the computer must be directed to store the data in a sequence compatible with the requirements of the program statements.

230 END

Directs the system to end the execution of the program.

Entering And Running The Program

The sequence of statements representing the problem and its solution can now be entered at the terminal. The complete program would appear as below, assuming no errors have been made. To run the program, the control command RUN is given.

```
*5   REM TOTAL MILES/GALS
*10  PRINT "OLD MILES ";"NEW MILES ";"MITR ";"GAL GAS ";"AMPG"
*20  PRINT"----------------------------------------------"
*30  READ L
*40  LET L1 = L
*50  READ N
*60  IF N=0 THEN 190
*70  READ G
*80  LET M=N-L
*90  IF M=0 THEN 12
*100 LET A=M/G
*110 IF A >< 0 THEN 130
*120 PRINT "YOUR TANK HAS A HOLE IN IT"
*130 IF A < 35 THEN 150
*140 PRINT "I DONT BELIEVE IT"
*150 PRINT L;N;M;G;A
*160 LET L=N
*170 LET G1=G1+G
*180 GO TO 50
*190 PRINT "TOTAL MILES/GALS",(L-L1)/G1
*200 DATA 3332,3553,14.8,3801,7.4,3926,15.2,4091,11.3,4275
*210 DATA 10.9,4460,9.8,4628,9.8,4864,12.3,5250,13.6,5617
*220 DATA 6.7,6112,10.0,6379,14.0,0
*230 END
*RUN

OLD MILES   NEW MILES   MITR   GAL GAS   AMPG
---------------------------------------------------------


   3332          3553      221       14.8      14.93243
   3553          3801      248        7.4      33.51351
   3801          3926      125       15.2      8.223684
   3926          4091      165       11.3      14.60177
   4091          4275      184       10.9      16.88073
   4275          4460      185        9.8      18.87755
   4460          4628      168        9.8      17.14286
   4628          4864      236       12.3      19.187
   4864          5250      386       13.6      28.38235
I DONT BELIEVE IT
   5250          5617      367        6.7      54.77612
I DONT BELIEVE IT
   5617          6112      495       10        49.50000
   6112          6379      267       14        19.07143
TOTAL MILES/GALS                  22.43741
```

Program With Loops

A program which creates a table of roots provides an opportunity to study the use of loops. In the following example, the range of numbers for which roots are desired are square root, cube root, and fourth root. The statement sequence entered at the terminal, and the resulting output are as follows:

```
*10 FOR X = 1 TO 15
*20 PRINT X,
*30 FOR R = 2 TO 4
*40 PRINT X**(1/R),
*50 NEXT R
*60 PRINT
*70 NEXT X
*80 END
*RUN
```

1	1	1	1
2	1.414214	1.259921	1.189207
3	1.732051	1.44225	1.316074
4	2	1.587401	1.414214
5	2.236068	1.709976	1.495349
6	2.44949	1.817121	1.565085
7	2.645751	1.912931	1.626577
8	2.8282427	2	1.681793
9	3	2.080084	1.732051
10	3.162278	2.154435	1.77828
11	3.316625	2.223980	1.821160
12	3.464102	2.289428	1.86121
13	3.605551	2.351335	1.898829
14	3.741657	2.410142	1.934336
15	3.872983	2.466212	1.96799

Statements 10 and 70 create the outer loop and determine the range of numbers. Statements 30 and 50 create the inner loop and determine the roots. Note the use of a comma (,) at the end of PRINT statements 20 and 40 to keep the output all on the same line, and the use of PRINT statement 60 to advance the output a line space after each execution of the inner loop and thus line the numbers up with their roots.

This brief program provides an indication of the power of loops by which hundreds of computations can be made by executing just a few statements, repeatedly.

Program With Subroutine

The following example is a program for determining the greatest common divisor (GCD) of three integers (using the Euclidean algorithm) and illustrates the use of subroutines.

```
*10 PRINT TAB(13);"A";TAB(28);"B";TAB(43);"C";TAB(58);"GCD"
*20 READ A,B,C
*30 LET X = A
*40 LET Y = B
*50 GOSUB 200
*60 LET X = G
*70 LET Y = C
*80 GOSUB 200
*90 PRINT A,B,C,G
*100 GOTO 20
*110 DATA 60,90,120
*120 DATA 38456,64872,98765
*130 DATA 32,384,72
*200 LET Q = INT(X/Y)
*210 LET R = X-Q*Y
*220 IF R = 0 THEN 300
*230 LET X = Y
*240 LET Y = R
*250 GOTO 200
*300 LET G = Y
*310 RETURN
*320 END
*RUN

             A              B              C              GCD
            60             90             120             30
         38456          64872          98765              1
            32            384             72              8

OUT OF DATA
```

Statement 20 assigns the values from the DATA statements to the variables A, B, and C. The first two numbers are selected in statements 30 and 40, and their GCD is determined in the subroutine statements 200 through 310.

The GCD just found is called X statement 60; the third number is called Y in statement 70; and the subroutine is entered from statement 80 to find the GCD of these two numbers. This number is, of course, the GCD of the three given numbers and is printed out with them, as directed by statement 90. Statement 100 tells the program to go back to statement 20 and process the next set of data. When all the DATA statement values have been used, the program will end with the message, "OUT OF DATA."

In this example, a different form of the PRINT statement PRINT TAB(X), is used. The TAB(X) function causes the next data field to be printed at the character position indicated by the value of X + 1, where X may be an expression such as 5*SINY.

Multiple TAB(X) functions may be used in the same PRINT statement, separated by semicolons (;) and interspersed with names of constants or variables as shown in the example.

Program With A List And Table

Below is a listing and the resulting output of a program which uses both a list and a table. The program computes the total sales of each of five salespersons all of whom sell the same three products. The list P gives the price per item of the three products, and the table S tells how many item of each product each person has sold. Product number 1 sells for $1.25 per item, number 2 for $4.30 per item, and number 3 for $2.50 per item; salesperson number 1 sold 40 items of the first product, 10 of the second, and 35 of the third, etc. The program reads in the price list via statements 10, 20, and 30, using data line 160, and reads the sales table via lines 40 through 80, using data in statements 170 through 190. (Statements 40 through 80 are nested loops.) The same program could be used again, modifying only statement 160 if the prices change, and only statements 170 through 190 to enter the sales for another month.

```
*5   DIM S(3,5),P(3)
*10  FOR I = 1 TO 3
*20  READ P(I)
*30  NEXT I
*40  FOR I = 1 TO 3
*50  FOR J = 1 TO 5
*60  READ S(I,J)
*70  NEXT J
*80  NEXT I
*90  FOR J = 1 TO 5
*100 LET S = 0
*110 FOR I = 1 TO 3
*120 LET S = S+P(I)*S(I,J)
*130 NEXT I
*140 PRINT "TOTAL SALES FOR SALESPERSON",J,"$",S
*150 NEXT J
*160 DATA 1.25, 4.30, 2.50
*170 DATA 40, 20, 37, 29, 42
*180 DATA 10, 16, 3, 21, 8
*190 DATA 35, 47, 29, 16, 33
*200 END

*RUN

TOTAL SALES FOR SALESPERSON    1    $    180.5
TOTAL SALES FOR SALESPERSON    2    $    211.3
TOTAL SALES FOR SALESPERSON    3    $    131.65
TOTAL SALES FOR SALESPERSON    4    $    166.55
TOTAL SALES FOR SALESPERSON    5    $    169.4
```

For the advanced programmer, forms of the PRINT statement and PRINT USING statement are available that permit more flexibility in the formatting of the program output.

Formatting Output With A Comma Or Semicolon

The end of a PRINT statement signals the end of the line, unless a comma or a semicolon is the last character of the statement.

For example, statement 20 in the program entry

```
*   10 FOR I = 1 TO 15
*   20 PRINT I
*   30 NEXT I
*   40 END
*   RUN
```

results in output of 15 numbers printed on 15 lines, thus:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

The use of a comma after a variable in a PRINT statement implies data placed in a zone format upon printout. BASIC provides for a line comprising five zones, each zone being referred to as a standard zone. By the use of a comma after a variable, data is allotted to zones and the data is right-justified within the zone. Thus, by rewriting statement 20 as

```
*   20 PRINT I,
```

The resulting format is

```
 1        2         3         4         5
 6        7         8         9        10
11       12        13        14        15
```

The statement

```
* 10 PRINT X, Y
```

results in the printing of the value of X in the first standard zone, the value of Y in the second standard zone and the return to the next line, while

```
* 20 PRINT X, Y,
```

results in the printing of these two values in the first and second standard zones and no return; the next value called for in a subsequent PRINT statement is printed in the third standard zone.

The statement

```
* 30 PRINT X, Y, Z, A, B, C
```

results in the printing of the first five values in the five standard zones across the page; the sixth value is printed in the first zone beneath the first value. Five values are the limit to a printout line, each value being restricted automatically within the confines of its zone upon printout. (Refer to the remarks of the PRINT statement.)

The use of a semicolon after a variable in a PRINT statement implies a variation of the standard zone format. Spacing is compacted to obtain more zones on the line. Minimum size zone is 7 columns and can contain a number up to 4 characters. The next larger size zone is 9 columns and contains up to 7 characters. All other fixed-point numbers are printed as 12-column zones. Negative numbers are preceded by a minus sign in the first column of a zone.

For the following program (note use of semicolon in statement 20), the printout of values would be in compacted zones as illustrated.

```
* 10 FOR I = 1 TO 15
* 20 PRINT I;
* 30 NEXT I
* 40 END
* RUN
   1    2    3    4    5    6    7    8    9    10    11    12
  13   14   15
```

Commas and semicolons can be used within the same PRINT statement. The statement

```
* 50 PRINT X,Y; Z,
```

results in the values of X and Z being printed in standard zones, while the zone of value Y would be compacted.

Text to be printed verbatim is referred to as a label. A label is printed just as it appears in the PRINT statement, left-justified in a zone. If two or more labels appearing in the same PRINT statement are comma-separated, the first label is printed left-justified in the first zone and each succeeding label is printed left-justified in the next succeeding available zone.

The statement

```
* 10 PRINT "X VALUE", "SIZE", "RESOLUTION"
```

results in the printout

```
X VALUE               SIZE                  RESOLUTION
```

Semicolon (or null-separated) labels in the same PRINT statement are printed with no character separation.

The statement

```
* 20 PRINT "OLD MILES"; "NEW MILES"
```

results in the printout

```
OLD MILESNEW MILES
```

If a label exceeds the length of a line, the line must be ended by quotation marks and its carryover on the next line or lines treated as additional PRINT statements.

Spacing Within An Output Line With Functions TAB(X) And SPC(X)

When used in a PRINT statement, functions TAB(X) and SPC(X) give the user additional control of spacing within an output line. These functions can be used as any field within the PRINT statement.

The TAB function is expressed in the form:

```
PRINT TAB (expression);    data to be printed
```

It causes the printing of the next data field at the character position indicated by the value of the expression plus one.

The SPC function is expressed in the form:

```
PRINT SPC (expression);    data to be printed
```

The number of spaces, equal to the value of the expression, is inserted in the print line. If this number causes the print position to exceed 72, the carriage returns and the print position indicator is set at 1.

Exception conditions:

TAB:

1. When the expression results in a number less than the current character position where the carriage is sitting, the TAB function is ignored.

2. When the expression results in a number greater than the line limit, the TAB function is ignored.

SPC:

When the expression results in a number which, when added to the current character position on the line, exceeds the line limit, the current line is printed and the current character position is reset to the first position on the next line.

Examples:

```
*  10 PRINT X, TAB(20); Y; TAB(40); Z
```

results in the values of X starting right-justified in the first zone, the values of Y starting at position 21, and the values of Z starting at position 41.

In the example

```
*  10 PRINT TAB(20); "DATA"
*  20 END
*  RUN
```

the resulting printout is DATA positioned as follows:

Position	20	21	22	23	24
		D	A	T	A

```
* 20 PRINT TAB(10*SIN(X)+10); X
```

results in the value of X being printed in the position specified by the value of the expression (10*SIN(X)+10).

In the example

```
*  10 FOR X = 1 TO 5
*  20 PRINT X; SPC(X); "+"
*  30 NEXT X
*  40 END
*  RUN
```

the resulting printout is

```
1 +                       (separated by 1 space)
2  +                                  (2 spaces)
3   +                                 (3 spaces)
4    +                                (4 spaces)
5     +                               (5 spaces)
```

Formatting Line Output

A line of output (a printed line) can be formatted by the user by means of the PRINT USING and PRINT # USING statements.

The fields that compose the image of the image statement pointed to by the PRINT USING and PRINT # USING statements can be made up of the following types:

integer	exponential	literal
decimal	alphanumeric	

Format control characters depict the fields within the image statement; the fields are separated by one or more literal characters (which may be blanks).

Each character following the colon of an "image" statement pointed to by a PRINT USING or PRINT # USING statement is treated as a print position, specifying either a literal or control character.

To facilitate explanation of format control characters and fields, the following examples make use of the PRINT USING statement only. The PRINT USING statement directs the system to immediately produce a visible result at the terminal upon program execution.

INTEGER-TYPE FIELD

Each numeric of an integer-type field is indicated by a number sign (#); the field width must also include a # for the algebraic sign, plus or minus. Upon program execution, the numbers of an integer type field are right-justified within the field and rounded if they are not integral.

Example:

```
*  10 LET A = 123
*  20 LET B = 12.34
*  30 PRINT USING 40,A,B
*  40:
*  50 END
*  RUN

     123  12
```

If a number does not fit into the specified format, a field of asterisks of the length specified is printed upon program execution.

Example:

```
*  10 LET A = 1234
*  20 PRINT USING 30,A,A
*  30: # #
*  40 END
*  RUN

     1234 ***
```

If an integer type field is preceded by a dollar sign ($), the $ "floats" up against the first nonzero digit in the field upon program execution.

Example:

```
*   10 LET A = 123
*   20 PRINT USING 30,A
*   30: $
*   40 END
*   RUN

       $123
```

DECIMAL-TYPE FIELD

Each numeric of a decimal-type field is indicated by a #; the field width must also include a # for the algebraic sign if minus. Upon program execution, the numbers of a decimal-type field are right-justified within the field and the value is rounded to the number of places specified by the #s following the decimal point.

Example:

```
*   10 LET A = 123.45
*   20 LET B = -3.456
*   30 LET C = -.017
*   40 PRINT USING 50,A,B,C
*   50:###.##  ##.####  #.##
*   60 END
*   RUN

       123.45   -3.4560   -.02
```

NOTE: The remarks concerning the use of the dollar sign and display of asterisks in regard to the integer type field also apply to the decimal type field.

EXPONENTIAL-TYPE FIELD

An exponential-type field is a decimal type field followed by four up-arrows (↑↑↑↑); the up-arrows serve to reserve space for placing an exponent. The field width must include a # for the algebraic sign if minus. For negative values, a minimum of two #s should be specified to the left of the decimal point to provide for the minus sign and at least one digit. The value is rounded as with decimal-type fields.

Example:

```
*  10 LET A = 123000000
*  20 LET B = 123.456
*  30 LET C = -.0177
*  40 PRINT USING 50,A,B,C
*  50:###.##↑↑↑↑ ##.####↑↑↑↑ #.###↑↑↑↑
*  60 END
*  RUN

     123.00E 06   1.2346E 02   -1.77E-02
```

ALPHANUMERIC TYPE FIELDS

An alphanumeric-type field can be specified in one of four possible ways, each indicated by the use of a single quote (') followed by one or more letters to indicate place of the alphanumeric string within the field. Note that the quote of the designated field is also a place holder. The fields are as follows:

'L...L indicates the string is to be left-justified within the field and blank-filled or truncated.

'R...R indicates the string is to be right-justified within the field and blank-filled or truncated.

'C...C indicates the string is to be centered within the field and blank-filled or truncated to the right. If an odd number of characters is to be centered in a specified format calling for an even number of characters, the string is centered one character to the left of a centered position.

'E...E indicates the string is to be left-justified within the field and the field is to be right-extended to accommodate the string if the string is longer than the field itself.

Example:

```
010  A$="ABCDEFG"
020  B$="ABCDEFGHIJKL"
030  PRINT"123456789012345678901234567890123456789012345678901234567890"
040  PRINT
050  PRINT USING 100,A$
060  PRINT USING 110,A$
070  PRINT USING 120,A$
080  PRINT USING 130,A$
090  PRINT USING 140,B$
100: 'LLLLLLLLL         LEFT JUSTIFIED IN A 10-CHAR FIELD
110: 'RRRRRRRRR         RIGHT JUSTIFIED IN A 10-CHAR FIELD
120: 'CCCCCCCCC         CENTER JUSTIFIED IN A 10-CHAR FIELD
130: 'EEEEEEEEE         EXTENDED FIELD LONGER THAN STRING
140: 'EEEEEEEEE         EXTENDED FIELD SHORTER THAN STRING
150  END
```

When executed, this program prints:

```
123456789012345678901234567890123456789012345678901234567890

ABCDEFG          LEFT JUSTIFIED IN A 10-CHAR FIELD
   ABCDEFG       RIGHT JUSTIFIED IN A 10-CHAR FIELD
 ABCDEFG         CENTER JUSTIFIED IN A 10-CHAR FIELD
ABCDEFG          EXTENDED FIELD LONGER THAN STRING
ABCDEFGHIJKL     EXTENDED FIELD SHORTER THAN STRING
```

LITERAL-TYPE FIELD

A literal-type field is composed of characters (other than control characters). Upon program execution, the field appears exactly as indicated by the image statement.

Example:

```
*  10 LET A = 123.450
*  20 PRINT USING 30,A
*  30: THE VALUE OF A IS  $.####.##
*  40 END
*  RUN

     THE VALUE OF A IS     $123.45
```

CONCATENATION OF MULTIPLE FORMATTED IMAGES

The output of multiple PRINT USING or PRINT # USING statements can be placed on one line by use of a comma or semicolon following an output list. Images are concatenated end-to-end. When used in conjunction with MARGIN to extend the rightmost character position, lines can be formatted beyond the normal length of 75 characters.

DEFINING FUNCTIONS

The user can define any function which is expected to be used a number of times in a program by use of a DEF statement. The name of the defined function must be three alpha characters. The user can define up to 26 functions. One suggested method of accounting for the number of functions within a program is to label function names alphabetically; e.g., FNA, FNB..., FNZ.

The value of such a function can be seen in a program where the user frequently needs the function (e raised to -x squared). The function would be introduced by the statement:

```
* 10 DEF FNE (X) = EXP (-X**2)
```

and later on called for various values of the function by such statements as

```
* 100 LET A = FNE(.1)
* 200 LET B = FNE(3.45)
```

Such a definition can be a great time-saver when the user wants values of some function for a number of different values of the variable.

The function to find the length of the hypotenuse of a right triangle serves as another example. Given sides of X and Y, the function can be formatted in the statement

```
* 20 DEF FNA(X,Y) = SQR (X**2 + Y**2)
```

The function can then be used in the program as often as desired. For example:

```
* 50 LET H = FNA(3,4)
* 60 LET G = FNA(A+6,B-3)
```

The PRINT statement

```
* 70 PRINT H,G
```

then results in the printout of the two required answers.

The DEF statement must occur previous to the use of the function in the program, and the expression to the right of the equal sign can be any formula that can fit onto one line. It can include any combination of other functions, including those defined by different DEF statements, and it can involve other variables besides the one denoted as the argument of the function. Thus, assuming FNR is defined by:

```
* 10 DEF FNR(X) = SQR (2+LOG(X)-EXP(Y*Z)*(X+SIN(2*Z)))
```

the current values of Y and Z are used in the computation of X.

A DEF statement can contain up to nine arguments; the total number of arguments for all DEF statements within a program is limited to 99.

If an argument is preceded by a sign, the argument with its sign must be enclosed in parentheses. For example, A=FNA(X,(-Y)).

MULTIPLE-LINE DEF STATEMENT

The user may want to use the DEF statement wherein he wishes to assign arguments or values that cause the statement to exceed the length of a line. If a DEF statement requires more than one line for the definition of a function, the function can be introduced with a DEF statement in which no equal sign appears, continue in a series of lines in which arguments or values are assigned, and end in a line containing the word FNEND. The function is thus defined in a multiple-line DEF statement, the end of the statement indicated by the line FNEND. Local variables defined within a function definition bear no relation to similarly named variables used outside the definition. Multiple-line DEF statements may not be nested. Transfers from inside a multiple-line DEF statement to outside, and vice versa, are not allowed.

The following examples illustrate the use of the multiple-line DEF statement.

Example 1

```
*10  DEF FNX(A,B)
*20  FNX=A
*30  IF A < B THEN 50
*40  FNX=B
*50  PRINT "FNX=";FNX
*60  FNEND
*70  X1=FNX(5,7)
*80  END
*RUN
```

Lines 10 through 60 constitute the DEF statement. The program results in the printout.

```
FNX=  5
```

Example 2

```
*10  C=3
*20  D=4
*30  DEF FNA(X,Y)C,D
*40  C=5
*50  D=10
*60  FNA=X
*70  IF X=Y THEN 90
*80  FNA=Y
*90  PRINT "C="C;"D="D;"FNA="FNA
*100 FNEND
*110 C1=FNA(9,7)
*120 PRINT"C="C;"D="D
*130 END
*RUN
```

Lines 30 through 100 constitute the DEF statement; therefore, the values of C and D outside the statement bear no relation to values of C and D assigned within the statement. The program results in the printout

```
C=  5  D=  10  FNA=  7
C=  3  D=   4
```

DATA INPUT DURING PROGRAM EXECUTION

There are times when it is desirable to enter data during the running of a program. This is particularly true when one person writes the program and saves it in the system and other persons are to supply the data when they wish to make use of the program. Data may be requested by means of an INPUT statement, which acts as a READ statement but does not draw numbers from a DATA statement. If, for example, the user is to supply values for X, Y, and Z into a program, the statement

```
INPUT X,Y,Z
```

appears ahead of the first statement that is to use these variables. When the system encounters this statement during program execution, the terminal prints out a question mark. The user then types values for X, Y, and Z immediately after the question mark, each separated by a comma, presses the return key, and the computer resumes program execution.

An INPUT statement can be used in conjunction with a PRINT statement to permit identification of variable values being requested. The user can employ the sequence

```
* 20 PRINT "WHAT ARE X, Y, Z";
* 30 INPUT X, Y, Z
```

and the terminal prints out the following during program execution:

```
WHAT ARE X, Y, Z?
```

to which the user must respond with values, on the same line. (Without the semicolon at the end of statement 20, the question mark would have been printed on the next line.)

If an INPUT statement is employed in a loop to repeatedly request input of a numeric value, program execution must be terminated by typing the letter S (or any word beginning with the letter S, e.g., STOP) after the question mark.

It may take a long time to enter large amounts of numeric values using INPUT statements. Therefore, INPUT statements should be used only when small quantities of values are to be entered, or when there is a requirement to enter values during the running of the program.

NOTE: The special case for matrix data input during program execution when use is made of the MAT INPUT statement is described in "Matrices" below.

A program to convert degrees Fahrenheit to Celsius serves to illustrate the usefulness of the INPUT statement. Because this program is designed to loop back to the program beginning each time to demand another input, the user must type in the word STOP after the question mark when the user wishes to terminate the program.

```
*  10 PRINT "FAHRENHEIT";
*  20 INPUT F
*  30 LET C = (F-32) * 5/9
*  40 PRINT "CELSIUS =" C
*  50 PRINT
*  60 GOTO 10
*  70 END
*  RUN

FAHRENHEIT ?32
CELSIUS = 0

FAHRENHEIT ?212
CELSIUS = 100

FAHRENHEIT ?STOP
```

MATRICES

A set of special statements is provided for operating upon matrices. These statements are identified by the word MAT, with which each such statement begins. Although the user can construct programs using only elementary BASIC to perform calculations on--or otherwise manipulate--matrices, the set of MAT statements simplifies the programming effort by shortening programs considerably.

The format of the MAT statements are:

MAT READ A,B,C,...	Read into matrices A, B, C,..., their dimensions having been previously specified. Data is read in row-wise sequence from standard-format DATA statements, and entered into the matrices. Each matrix may be totally or partially filled. Zeroes are automatically assigned to any unfilled positions.
MAT PRINT A,B,C,...	Print matrices A, B, C,... The semicolon, TAB, and SPC can be used, as in the normal PRINT statement. Double space is provided for between rows; between folded parts of the same row, single space is provided.
MAT INPUT A	Input desired values for elements of matrix A during program execution time.
MAT C = A + B	Add two matrices A and B and store result in matrix C.
MAT C = A - B	Subtract matrix B from matrix A and store result in matrix C.
MAT C = A * B	Multiply matrix A by matrix B and store result in matrix C.
MAT C = INV(A)	Invert matrix A and store resulting matrix in C.
MAT C = TRN(A)	Transpose (interchange rows and columns) matrix A and store resulting matrix in C.

MAT C = (K) * A or MAT C = A * (K)	Multiply matrix A by value represented by K. K may be either a number or an expression, but in either case it must be enclosed in parentheses.
MAT C = CON	Each element of matrix C is set to one.
MAT C = ZER	Each element of matrix C is set to zero.
MAT C = IDN	Diagonal elements of matrix C are set to one's, yielding an identity matrix.

The last three MAT statements can also be written with subscripts suffixed to the right-hand side; e.g., MAT C = ZER(I,J). The use of this form is described below.

Special rules apply to the dimensioning of matrices which occur in MAT instructions. DIM statements indicate the maximum dimension of a matrix. Thus

```
DIM M(20,35)
```

means that M can have up to 20 rows and up to 35 columns. The dimensions of all matrices occurring in MAT statements must be specified in DIM statements; otherwise, automatic dimensioning (subscript values of 10 or less) is implied.

NOTE: Rows and columns of matrices are numbered 1 through n. That is, there is no row or column numbered 0 in matrices used in MAT statements.

The current dimension of a matrix can be determined either when it is initially defined by the dimension statement or by special usage of certain MAT statement forms. The four general forms used to accomplish dynamic redimensioning are:

1. MAT READ A(M,N)
2. MAT A = ZER(M,N)
3. MAT A = CON(M,N)
4. MAT A = IDN(N,N)

The first, MAT READ, redefines the current dimensions of matrix A as M rows and N columns and then reads M*N data values to fill in the elements. More than one matrix may be redimensioned and read with a single statement.

The other three forms are used to redefine the current dimensions of a matrix (A) and then fill its elements with values as specified by the statement type.

The rules for dynamic redimensioning are as follows:

1. No dimension can be changed to a value that exceeds its original declaration in the DIM statement.

2. Using the statement types described above, dimensions can be redefined in either the upward or downward direction as long as the definition is within the bounds of item 1 above and the original declaration in the DIM statement.

For example, a matrix specified in the DIM statement as (6,4) might be redimensioned as (4,4), but not as (10,2) -- by rule 1 -- or (5,5) -- by rule 2.

In addition to use of a DIM statement, and possibly a declaration of current dimensions, the user must use MAT statements with care. For example, a matrix product MAT C = A * B may be illegal for one of two reasons: A and B may have dimensions such that the product is not mathematically defined, or even if it is defined, C may not have reserved enough space for the answer. In either case, the message IN XXXX DIM ERROR results, where XXXX is the line number of the statement in question.

The same matrix can occur on both sides of a MAT statement in cases of addition, subtraction, multiplication by a constant, or inversion, but not in any other statement form. Legal form are:

```
MAT A = A + B
MAT A = A - B
MAT A = (2.5)*A
MAT A = INV (A)
```

Also, note that the special form of matrix multiplication

```
MAT B = A * A
```

is legal.

Illegal forms are:

```
MAT A = B
MAT A = B*A
MAT A = TRN(A)
MAT A = A + B - C
```

The last example is an attempt to use more than one arithmetic operator in a MAT statement. Each matrix operation requires its own matrix statement.

A 2-dimensional string matrix (e.g., A1$(10,20)) is not permitted. No MAT operations are permitted for string variables.

The following program illustrates some simple operations upon matrices by the use of MAT statements.

```
*   10  DIM A(2,3), B(2,3), S(2,3)
*   20  DIM D(2,3), M(2,3), T(3,2)
*   30  MAT READ A,B
*   40  REM SUM OF MATRICES
*   50  MAT S = A + B
*   60  MAT PRINT S
*   70  REM DIFFERENCE OF MATRICES
*   80  MAT D = A - B
*   90  MAT PRINT D
*   100 REM MULTIPLY MATRIX
*   110 MAT M = (2) * A
*   120 MAT PRINT M
*   130 REM TRANSPOSE MATRIX
*   140 MAT T = TRN(B)
*   150 MAT PRINT T
*   160 DATA 1,2,3,4,5,6
*   170 DATA 6,5,4,3,2,1
*   180 END
*   RUN
 7            7            7

 7            7            7


-5           -3           -1

 1            3            5


 2            4            6

 8           10           12


 6            3

 5            2

 4            1
```

The MAT INPUT statement permits input of data, pertaining to the elements of a matrix, at program execution time. The function NUM(X) can be utilized to supply a count of the number of data elements entered; thus, the matrix array can be filled to any level desired (.i.e., user need not input data elements to fill the entire array). The count of NUM(X) always reflects the number of input data elements for the most recently executed MAT INPUT statement. If more than one line of values is required, the line (and subsequent lines, if needed) is terminated with an ampersand (&) to indicate continuation. The ampersand may or may not be comma-separated from the last value. The MAT INPUT statement can be used with either 1- or 2-dimensional arrays. The 1-dimensional array is filled beginning with element 1; 2-dimensional arrays are filled in a row sequence.

Two examples of the use of the MAT INPUT statement are as follows:

Example 1:

```
*  10 DIM S(100)
*  20 MAT INPUT S
*  30 PRINT S(1);" + ";S(2);" = ";S(1)+S(2)
*  40 LET T = S(1)+S(2)
*  50 FOR I = 3 TO NUM(X)
*  60 LET T = T + S(I)
*  70 PRINT"        + ";S(I);"  =  ";T
*  80 NEXT I
*  90 END
*RUN

?1,2,3,4,5,&
?6,7,8,&
?9,10,11
    1   +     2    =    3
        +     3    =    6
        +     4    =   10
        +     5    =   15
        +     6    =   21
        +     7    =   28
        +     8    =   36
        +     9    =   45
        +    10    =   55
        +    11    =   66
```

Example 2:

```
*  10 DIM M1(3,4)
*  20 MAT INPUT M1
*  30 MAT PRINT M1;
*  40 END
*RUN

?1,2,3,4,5,6,7

    1    2    3    4
    5    6    7    0
    0    0    0    0
```

ADDITIONAL FUNCTIONS

BASIC provides for the use of other functions in addition to the standard mathematical functions listed in Section IV.

These additional functions are as follows:

INT(X)	TIM(X)	NUM(X)	TAB(X)	LEN(X$)	STR$(N)
RND(X)	CLK$	SST(X$,Y,Z)	SPC(X)	LIN(X)	VAL(S$)
SGN(X)	DAT$			ASC(X)	TST(S$)
DET(X)					HPS(X)

Function INT(X)

Purpose: To truncate a number to integer form.

Format: INT (expression)

Examples:
```
* 10 PRINT INT (2.35)
* 20 PRINT INT (-2.35)
* 30 PRINT INT (2.9)
```

are three examples of this function placed in a PRINT statement and used to truncate a number. The resultant printouts would produce 2, -3, and 2, respectively.

Function RND(X)

Purpose: To generate random numbers for computational procedures requiring random variables.

Format: The general format is

RND (any variable or constant)

which produces a random number between (but not including) 0 and 1.

If a great number of these random numbers are produced, it becomes apparent that they tend to fall uniformly over the range, for the numbers come from a uniformly distributed population.

Examples:
```
* 10 FOR L = 1 TO 20
* 20 PRINT RND(X),
* 30 NEXT L
* 40 END
* RUN
```

might generate the following:

0.3199251	0.0590169	0.4018556	0.6280534	0.2292995
0.8075665	0.964758	0.2424602	0.066037	0.368314
0.3074467	0.4493044	0.7489442	0.4024822	0.301177
0.7088735	0.2340001	0.9746831	0.5227955	0.6405085

If random integers between 0 and 9 are desired, statement 20 can be changed to read

```
* 20 PRINT INT (10*RND(X)),
```

which results in

3	0	4	6	2
8	9	2	0	3
3	4	7	4	3
7	2	9	5	6

If statement 20 were changed to read

```
*  20 PRINT INT (20*RND(X)+5),
```

then the printout would contain random numbers between integers 5 and 24.

The range of random numbers generated, therefore, is dependent upon how function RND(X) is modified.

The function RND(X) lends itself readily to programs involving probability. For example, to simulate a 5-trial coin tossing contest, the following program can be written:

```
*  10 FOR T = 1 TO 5
*  20 IF RND(T)  =0.5 THEN 50
*  30 PRINT "HEADS"
*  40 GOTO 60
*  50 PRINT "TAILS"
*  60 NEXT T
*  70 END
```

The program execution is a reasonable facsimile of the results of a coin tossed five times.

The use of the RND function as described above is appropriate when the same sequence of random numbers is to be generated each time a program is run. If the variable or constant used as an argument is a positive quantity and is not changed, the same sequence of random numbers is generated for each execution of the program.

The use of a negative argument for the RND function causes an unpredictable series of random numbers to be generated each time the program is run. For example, if the user wishes different sequences of random numbers for each execution of his program, one of the following techniques can be used:

```
*  10 LET X = -1
*  20 FOR I = 1 TO 20
*  30 PRINT RND(X)
*  40 NEXT I
*  50 END

*  10 LET X = 1
*  20 FOR I = 1 TO 20
*  30 PRINT RND(-X)
*  40 NEXT I
*  50 END

*  10 FOR I = 1 TO 20
*  20 PRINT RND(-1)
*  30 NEXT I
*  40 END
```

Function SGN(X)

Purpose: To determine the sign of an expression.

Format: SGN (expression)

The function yields +1, -1, or 0, depending upon the value of the expression. The following list gives the options:

	(Value of expression)	Yields
SGN	(zero)	0
SGN	(positive,non-zero)	+1
SGN	(negative,non-zero)	-1

Examples:

```
* 10 IF SGN(X) = 1 THEN 100
```

In this statement, the value of X must be positive to accomplish the transfer of processing to statement 100.

The statement

```
* 20 LET X = SGN(Y)*ABS(X)
```

assigns to X the sign resulting from the value of Y.

Function DET(X)

Purpose: To obtain the determinant of the last matrix inverted.

Format: DET (any variable or constant)

Examples:

```
*10  MAT B=INV(A)
*20  LET C=DET(X)
*30  PRINT C
```

The program, when executed, inverts matrix A, stores the result in matrix B, and prints out the value of C, the determinant of matrix A.

The determinant can be made an element of a more complex numeric expression.

```
*10  PRINT 2*DET(X)
*20  IF DET(X)=0 THEN 60
```

Any attempt to invert a singular matrix does not stop the program, but DET(X) is set to zero. For any program, the user must decide if a determinant is large enough to be meaningful.

Function TIM(X)

Purpose: To obtain elapsed processor time in seconds.

Format: TIM (any keyboard character)

Examples: *50 PRINT "PROCESSOR TIME=";TIM(X);"SECONDS"

A program including such a statement, when executed, would contain a printout line

```
PROCESSOR TIME= <value> SECONDS
```

The processor time can be assigned a variable name.

```
*50  LET T=TIM(X)
*60  PRINT "PROCESSOR TIME =";T
```

Function CLK$

Purpose: To provide the time of day as a string.

Format: CLK$

Examples: *50 PRINT CLK$

A program including such a statement, when executed, would contain a printout line indicating time of day in hours ranging from 1 to 24 and in portions of hours, such as NN.NNN.

The time of day can also be assigned to a string variable.

```
*10 LET T$=CLK$
*20 PRINT T$
```

Function DAT$

Purpose: To provide the calendar date as a string.

Format: DAT$

Examples: *50 PRINT DAT$

A program including such a statement, when executed, would contain a printout line indicating the calendar date (month, date, year), such as

```
MM/DD/YY
```

The calendar date can also be assigned to a string variable.

```
*10 LET A$=DAT$
*20 PRINT A$
```

Function NUM(X)

Purpose: To supply count of number of data elements in response to request from MAT INPUT statement.

Format: NUM (any alphanumeric character)

Refer to MAT INPUT statement under "Matrices" in this section for an example concerning use of NUM(X).

Function SST(X$,Y,Z)

Purpose: To extract selected characters of a string.

Format: SST(string variable, beginning character, number of characters)

Refer to the use of the LET statement under "Alphanumeric Data and String Manipulation," in this section, for an explanation of the use of this function.

Function TAB(X)

Purpose: To position data field at indicated character position within an output line.

Format: TAB(expression), <data to be printed>

Refer to "Spacing Within An Output Line with Functions TAB(X) and SPC(X)," in this section, for an explanation of the use of this function.

Function SPC(X)

Purpose: To insert spaces at indicated positions within an output line.

Format: SPC(expression); <data to be printed>

Refer to "Spacing Within an Output Line with Functions TAB(X) and SPC(X)," in this section, for an explanation of the use of this function.

Function LEN(X$)

Purpose: To determine the number of characters in a specified string variable.

Format: LEN(string variable)

Examples:

```
*   10 READ A$,B$,C$
*   20 PRINT LEN(A$);LEN(B$);LEN(C$)
*   30 DATA LENGTH, OF, STRING
*   40 END
*   RUN
```

results in a printout of

```
  6   2   6
```

The value of LEN can be assigned to a variable.

```
*   10 LET X=LEN(A$)
*   20 PRINT"LENGTH OF STRING=";X
```

Function LIN(X)

Purpose: To provide the last line number encountered in reading from or writing to a file.

Format: LIN(file designator)

Examples:

```
*    10 FILES A
*    20 SCRATCH #1
*    30 FOR I=1 to 45
*    40 WRITE #1,I;
*    50 NEXT I
*    60 PRINT "LAST LINE WRITTEN IS ";LIN(1)
*    70 RESTORE #1
*    80 PRINT
*    90 FOR I=1 to 24
*   100 READ #1,X1
*   110 PRINT X1;
*   120 NEXT I
*   130 PRINT
*   140 PRINT "LAST LINE READ IS ";LIN(1)
*   150 END
*   RUN
```

upon execution, the program produces

```
LAST LINE WRITTEN IS 50

 1   2   3   4   5   6   7   8   9  10  11  12
13  14  15  16  17  18  19  20  21  22  23  24

LAST LINE READ IS 30
```

The listing of file A shows that it contains the following data:

```
000010  1,   2,   3,   4,   5,   6,   7,   8,   9,
000020  10,  11,  12,  13,  14,  15,  16,  17,  18,
000030  19,  20,  21,  22,  23,  24,  25,  26,  27,
000040  28,  29,  30,  31,  32,  33,  34,  35,  36,
000050  37,  38,  39,  40,  41,  42,  43,  44,  45,
```

The value of LIN can be assigned to a variable.

```
*    10 LET N=LIN(1)
*    20 PRINT "LAST LINE READ IS ";N
```

Function ASC(X)

Purpose: To provide the numeric value of a specified character or, for the case of non-printing characters, an abbreviation.

Format: ASC $\left\{ \begin{array}{l} \text{(character)} \\ \text{(abbreviation)} \end{array} \right\}$

Examples:

```
*    10 PRINT "VALUE FOR A IS ";ASC(A)
*    20 PRINT "VALUE FOR CR IS ";ASC(CR)
*    30 END
*    RUN
```

results in

```
VALUE FOR A IS 65
VALUE FOR CR is 13
```

The value of ASC can be assigned to a variable.

```
*    10 LET X=ASC(A)
*    20 PRINT "VALUE FOR A IS ";X
```

The conversion equivalents for characters and non-printing characters are listed in the table "Numeric Code Table" in this section.

Function STR$(N)

Purpose: To produce a string corresponding to a value of a number represented by an expression.

Format: STR$ (expression)

Examples: The value of STR$ can be assigned to a string variable

```
*10  LET X$=STR$(N)
```

or can be used directly

```
*20  PRINT STR$(N)
```

where N is a number, STR$ converts N to a string containing the same digits.

```
*10  LET N=77.233
*20  LET X$=STR$(N)
*30  LET Y$=STR$(63)
*40  PRINT X$;Y$
*50  END
```

when executed, the program results in

```
77.233     63
```

Use of STR$ implies placement of the string right-justified, followed by a blank, in the smallest zone into which it will fit. Blanks occupy the remaining character positions of the zone.

Function VAL (S$)

Purpose: To produce a numeric value corresponding to the value of a string represented by a string variable.

Format: VAL (string variable)

Examples: The value of VAL can be assigned to a variable

```
*10  LET A=VAL(S$)
```

or can be used as an element of a numeric expression

```
*20  LET A1=2*VAL(S$)
*30  PRINT 3*VAL(S$)+A+A1
```

The string variable of VAL must be a valid constant. The program

```
*10  LET A$="12345"
*20  LET B$="12.95"
*30  LET C=VAL(A$)
*40  PRINT C;VAL(B$)
*50  END
```

when executed, results in

```
12345      12.95
```

Function TST(S$)

Purpose: To produce a 1 as output if a string represented by a string variable can be interpreted as a number, or produce a 0 if the string cannot be interpreted as a number.

Format: TST (string variable)

Examples: The value of TST can be assigned to a variable

```
*10  LET T=TST(S$)
```

or can be used as an element of a numeric expression

```
*20  PRINT VAL(S$)*TST(S$)
*40  IF TST(S$)=0 THEN 50
```

The program

```
*10  LET A$="49"
*20  LET T=TST(A$)
*30  IF T=0 THEN 50
*40  PRINT VAL(A$)
*50  END
```

when executed, results in

```
49
```

Function HPS(X)

Purpose: To provide a horizontal print position of the next field to be transmitted to a specified file.

Format: HPS (file designator)

Examples: The function can be assigned to a variable

```
*10  LET P=HPS(0)
```

or can be used as an element of a numeric expression

```
*20  PRINT 12+HPS(0)
```

The program

```
*10  FOR X=1 to 8
*20  PRINT X;
*30  NEXT X
*40  LET A=HPS(0)
*50  PRINT A
*60  END
```

when executed, results in

```
1  2  3  4  5  6  7  8  49
```

The horizontal print position of the file is 49.

```
*10  FILES OUT1
*20  SCRATCH #1
*30  FOR I=1 TO 5
*40  WRITE #1,I;
*50  NEXT I
*60  PRINT "HOR. PRINT POS. OF FILE 1=";HPS(1)
*70  END
```

This program when executed, results in

```
HOR. PRINT POS. OF FILE 1=44
```

A listing of file OUT1 would show

```
10   1, 2, 3, 4, 5,
```

The file designator for function HPS must be a numeric value between zero and 8 inclusive. Zero is interpreted as being the user's terminal.

The use of function HPS is limited to providing the horizontal print position for output. If the specified file is open for input, a zero horizontal print position is returned.

SUBROUTINES

When a particular part of a program is to be performed more than one time, or possibly at several different places in the overall program, the part or parts are most efficiently programmed as subroutines. Subroutines can be likened to programs within the main program which permit the user to partition his main program.

The subroutine is entered by the way of a GOSUB statement. For example,

```
*  90 GOSUB 210
```

directs the processing to jump to statement 210, the first statement of the subroutine. The last statement of the subroutine to be executed must be a RETURN statement directing the processing to return to the earlier part of the program. For example,

```
*  350 RETURN
```

tells the processing to go back to the first statement numbered greater than 90 and to continue the program from there.

GOSUB statements can be used within subroutines to branch to still other subroutines. The following nonsense program illustrates the technique:

```
*    10 READ L
*    20 GOSUB 50
*    30 PRINT A,B,C,
*    40 STOP
*    50 REM THIS IS SUBROUTINE 1
*    60 LET A = 5
*    70 GOSUB 100
*    80 LET B = 10
*    90 RETURN
*   100 REM THIS IS SUBROUTINE 2
*   110 LET C = 15
*   120 FOR I = 1 TO L
*   130 LET C = I*C
*   140 NEXT I
*   150 RETURN
*   160 DATA 5
*   170 END
```

Statement 20 jumps the processing to Subroutine 1. Statement 70, in turn, transfers processing from Subroutine 1 to Subroutine 2. Statement 150 then returns the processing to the most recent point of departure -- statement 80. When statement 90 is encountered, processing is returned to statement 30. Statement 40 prevents the program from falling back into Subroutine 1 again and the program is terminated.

LOOPS

Frequently, there are operations in programming that must be repeated many times; therefore, some statements within a program must be executed many times. This repetition of a set of statements is referred to as a loop. For example, if a table were required of the first 100 positive integers and their square roots, it could be obtained by this program.

```
*     10 PRINT 1, SQR(1)
*     20 PRINT 2, SQR(2)
       .      .     .
       .      .     .
       .      .     .
*    990 PRINT 99, SQR(99)
*   1000 PRINT 100, SQR(100)
*   1010 END
```

By means of two BASIC statements, a programming loop can be written that accomplishes the same as the program with 101 statements but in only four statements; namely,

```
*   10 FOR X = 1 TO 100
*   20 PRINT X, SQR(X)
*   30 NEXT X
*   40 END
```

The FOR statement denotes the beginning of the loop, and it specifies the range (1 to 100) for the given variable (X) and in unit steps (implied step-size of 1 when STEP is not given) as the program keeps passing through the loop. If the steps were to be increments of other than 1, then statement 10 would include the word STEP followed by the required size. If the increments were, say 2, then the statement would be written as

```
* 10 FOR X = 1 TO 100 STEP 2
```

The NEXT statement (statement 30) terminates the loop and returns the system to statement 10, with the statement between being executed for each pass through the loop. When the loop has been executed the specified number of times (100, in the example), then it directs the system to the statement after the NEXT statement (statement 40).

The program loop described above is a simple one. The FOR and NEXT statements can be used effectively in more complex problems wherever iterations are required. For example, if integration of a function is required, the FOR statement can be used to define limits and set the count of iterations through the loop. Computation statements can then be made and the NEXT statement used to repeat the iteration until the count has been achieved.

It is possible, as well as useful, to have loops within loops. However, a loop cannot cross another loop. To illustrate:

This method of creating loops is allowed:

```
┌──── FOR X
│       .
│       .
│       .
│  ┌──FOR Y          For each pass through the X loop,
│  │    .
│  │    .            the Y loop is executed the
│  │    .
│  └──NEXT Y         specified number of times.
│       .
│       .
│       .
└─────NEXT X
```

For example, if the X loop had a range of 5 and the Y loop a range of 10, then for each pass through the X loop, the Y loop is executed 10 times. When the X loop has been executed 5 times, the Y loop will have been executed 50 times (i.e., 10 Y passes per 1 X pass).

This method is also allowed:

```
┌──FOR X
│┌─FOR Y
││┌FOR Z
││└NEXT Z
││┌FOR W
││└NEXT W
│└─NEXT Y
│ ┌FOR Z
│ └NEXT Z
└──NEXT X
    .
    .
    .
   END
```

This method is not allowed; note the cross-over of the loops:

```
┌─FOR X
│┌FOR Y
└┼NEXT X
 └NEXT Y
```

Loops can also be created within a program by the use of GOTO and READ statements. If a READ statement contains a variable to which the user wishes to assign more than one value, a GOTO statement directs the program to loop back to the READ statement and assign another value.

The loop is performed as many times as there are values available in a DATA statement. When the values have all been assigned, execution of the program is terminated and the message OUT OF DATA is printed.

The following sample program illustrates the use of a GOTO-READ loop:

```
10 READ A,B,D,E
15 LET G = A*E-B*D
20 IF G = 0 THEN 65
30 READ C,F
37 LET X = (C*E-B*F)/G
42 LET Y = (A*F-C*D)/G
55 PRINT X,Y
60 GOTO 30
65 PRINT "NO UNIQUE SOLUTION"
70 DATA 1,2,4
80 DATA 2,-7,5
85 DATA 1,3,4,-7
90 END
```

This program has assigned one set of values to the variables A, B, D, E, but three values each to the variables C and F. Therefore, the solution should provide six answers. To achieve multiple answers, a loop is created by way of statement 60. Here the program is directed back to statement 30 to assign new values to C and F from the data block.

The program and the resulting run would appear as follows:

```
*  10 READ A,B,D,E
*  15 LET G=A*E-B*D
*  20 IF G = 0 THEN 65
*  30 READ C,F
*  37 LET X = (C*E-B*F)/G
*  42 LET Y = (A*F-C*D)/G
*  55 PRINT X,Y
*  60 GOTO 30
*  65 PRINT "NO UNIQUE SOLUTION"
*  70 DATA 1,2,4
*  80 DATA 2,-7,5
*  85 DATA 1,3,4,-7
*  90 END
*  RUN

           4          -5.5
   0.6666667     0.1666667
   -3.666667     3.8333333

OUT OF DATA IN 30
```

LISTS AND TABLES

Often when writing a program, the need arises to make use of a list of numbers. The user will find it most advantageous to give the list a single variable name rather than provide separate variables for each number in the list. For example, if 25 salesmen were to be listed in a program, the list could be called S and the salesmen would be represented by S and a subscript, ranging from S(1) to S(25). Thus S(5) would represent the fifth salesman in list S and S(25) would represent the 25th or last salesman in the list.

The user may also find the need to make use of tables in his programs. Here again, a single variable name rather than separate variables for each entry of a table is most convenient. For example, P(3,J) would represent row 3, column J in table P; table P could be a 5 by 10 array. P(5,10) represents the entire table and could be dimensioned as such in a DIM statement.

Lists and tables thus permit the user to enter groups of numbers into a program that are to be worked upon concurrently. Such programs can be used over and over again, with the user updating the data each time the program is used.

The usefulness of employing a list in a program can be illustrated by an example. A brush salesman has 10 kinds of brushes he carries in his sample case. At the end of the day, he wishes to compute the dollar value of the orders he has taken. The prices of the 10 brushes are as follows:

0.50, 1.75, 2.25, 2.75, 3.45, 4.00, 4.25, 4.75, 5.00, 5.25

In writing his program, the salesman enters his quantity of sales for individual brushes and then asks for a printout of total sales.

```
*    10 DIM P(10)
*    20 FOR I = 1 TO 10
*    30 READ P (I)
*    40 NEXT I
*    50 LET S = 0
*    55 FOR I = 1 TO 10
*    60 READ B
*    70 LET S = S + B * P(I)
*    75 NEXT I
*    80 PRINT "TOTAL SALES = $" S
*    90 DATA 0.50, 1.75, 2.25, 2.75, 3.45
*   100 DATA 4.00, 4.25, 4.75, 5.00, 5.25
*   110 DATA 0,5,7,3,12
*   120 DATA 25,15,30,10,35
*   130 END
```

At the end of each work day, the salesman updates DATA statements 110 and 120 to reflect his orders and obtain new sales totals.

Below is a listing and run of a program that uses both a list and a table.

```
*   5 DIM S(3,5),P(3)
*  10 FOR I = 1 TO 3
*  20 READ P(I)
*  30 NEXT I
*  40 FOR I = 1 TO 3
*  50 FOR J = 1 TO 5
*  60 READ S(I,J)
*  70 NEXT J
*  80 NEXT I
*  90 for J = 1 TO 5
* 100 LET S = 0
* 110 FOR I = 1 TO 3
* 120 LET S = S + P(I)*S(I,J)
* 130 NEXT I
* 140 PRINT "TOTAL SALES FOR SALESMAN",J,"$",S
* 150 NEXT J
* 160 DATA 1.25,4.30,2.50
* 170 DATA 40,20,37,29,42
* 180 DATA 10,16,3,21,8
* 190 DATA 35,47,29,16,33
* 200 END

*RUN

TOTAL SALES FOR SALESMAN    1    $    180.5
TOTAL SALES FOR SALESMAN    2    $    211.3
TOTAL SALES FOR SALESMAN    3    $   131.65
TOTAL SALES FOR SALESMAN    4    $   166.55
TOTAL SALES FOR SALESMAN    5    $    169.4
```

The user should be aware of the need to dimension a list or table to at least the minimum of the subscript value. But it may be expedient to dimension somewhat generously over the minimum to permit changes to an existing program. For example, the brush salesman would do well to change statement 10 in his program to:

```
*   10 DIM P(25)
```

This will enable him to use his program if he adds up to 15 additional kinds of brushes to his line.

Extra large dimensions can be defined in DIM statements (with a maximum of about 100,000 elements), but space in computers is limited and a realistic dimension is in the best interest of all users of the time sharing system.

ALPHANUMERIC DATA AND STRING MANIPULATION

BASIC has the ability to manipulate alphanumeric information in addition to numeric data. Data consisting of alphanumerics and certain special characters can be treated as if it were numeric data.

A sequence of alphanumeric data is referred to as a "string;" the string size, in turn, is limited to 132 valid characters. Initially, space for 20 characters is allocated; the space is then expanded if space for more characters is required. Manipulation of a string is by means of a "string variable," created by following any permissible BASIC variable with the character **$**. For example,

```
A$,K1$,X5$
```

are valid string variables. Manipulation, incidentally, should not be interpreted as meaning arithmetic operations; such operations cannot be performed on string variables.

The use of alphanumeric data and string manipulation are restricted to certain BASIC statements. The following is a list of these statements, each accompanied by explanation of alphanumeric data use and string manipulation as applicable. The use of quotes to enclose strings is recommended where doubt exists as to their use; superfluous quotes will be ignored by the system.

- DIM

  A user may set up a list of allied strings as a one-dimensional array. The DIM statement must then be used to reserve space. For example,

  ```
  * 10 DIM A$(15),B$(25)
  ```

  Space for fifteen 20-character strings are then reserved by **A$** and twenty-five 20-character strings by **B$**. The user may then select particular strings within a string list; for example, **A$**(4) would be the fourth string in the **A$** list and B$(6) the sixth string in the B$ list.

- LET

  The LET statement can be used to assign the contents of one string variable to another string variable, assign a string constant to a string variable, concatenate strings, and extract selected characters of a string. Quotes must enclose any assigned string constant. An ampersand (&) is used to indicate string concatenation.

The statement

```
*10 LET R$=T$
```

assigns the contents of the string T$ to R$.

The statement

```
*10 LET G$ = "THIS IS A STRING"
```

assigns the string, THIS IS A STRING, to G$.

String concatenation is limited within one LET statement to two string variables or one string constant and one string variable.

The statements

```
*10 LET A$ = "JOHN DOE "
*20 LET B$ = "EMPLOYEE NUMBER 12345"
*30 LET C$ = A$ & B$
*40 PRINT C$
```

or

```
*10 LET A$ = "JOHN DOE "
*20 LET C$ = A$   "EMPLOYEE NUMBER 12345"
*30 PRINT C$
```

when executed, produces the printout

```
JOHN DOE EMPLOYEE NUMBER 12345
```

Extraction of selected characters of a string is achieved by use of the substring extraction function, which has the general format

SST (string variable, beginning character, number of characters)

Where:

1. String variable is assigned contents of a string

2. Beginning character is numeric value to indicate position of character with which to begin extraction

3. Number value of characters to extract

Character positions of a string are numbered from left to right, 1 through 132. Based on three arguments supplied to the SST function, a substring is extracted and stored left-justified in the string variable specified to the left of the equal sign of the LET statement. Blanks within a string, of course, are considered as characters when the character count is made.

The statements

```
*10 LET A$ = "THIS IS A DEMONSTRATION OF THE SUBSTRING FUNCTION"
*20 LET B$ = SST(A$,1,10)
*30 LET C$ = SST(A$,11,14)
*40 LET D$ = SST(A$,25,25)
*50 PRINT B$
*60 PRINT C$
*70 PRINT D$
```

upon program execution, produces printouts of

```
THIS IS A
DEMONSTRATION
OF THE SUBSTRING FUNCTION
```

o IF-----THEN or IF-----GOTO

Strings and string variables can be manipulated with these statements also. Only one string variable is permitted on each side of the relational symbol and the string must be enclosed by quotes. Relational symbols indicate relation in regard to alphabetic order.

Examples are as follows:

```
*  10 IF G$ = "THIS IS A STRING" THEN 30

*  10 IF G$  >H$ GOTO 30

*  10 IF "MAY"< > M$ THEN 30
```

o CHANGE

The CHANGE statement can be used to convert ASCII characters (or strings) to a numeric list containing their equivalent (decimal) codes or vice versa.

The process involves two lists, one numeric, the other a string variable. When converting numeric codes to a character string, the numeric list is to contain the valid numeric equivalent of a single character in each element. Given the desired number of items to convert, the CHANGE command performs the conversion and concatenate the resulting characters into the string variable. The zero-th location in the list contains the number of characters in the string.

In changing from a character string, the command stores the related numeric code for each character into the elements of the numeric array.

Examples are as follows:

String A$:		%	A	6	+	D
List N(X):	5	37	65	54	43	68
X:	(0)	(1)	(2)	(3)	(4)	(5)

The program

```
*  10 A$ = "%A6+D"
*  20 CHANGE A$ TO N
*  30 FOR J=0 TO 10
*  40 PRINT N(J);
*  50 NEXT J
*  60 END
```

when executed, results in a printout of

```
5   37   65   54   43   68   0   0   0   0   0
```

indicating that the operation has assigned five characters to the list and has entered the numeric code equivalents of these five alphanumeric characters into locations 1 through 5 in the list. The remaining locations contain zeroes.

Note that lists are automatically dimensioned at 10 unless otherwise specified. If the string to be changed contains more than 10 characters, a statement DIM N(X) must be provided prior to the CHANGE statement, where X is equal to or greater than the number of characters in the string.

The program

```
*  10 N(0)=5
*  20 N(1)=37 N(2)=65 N(3)=54 N(4)=43 N(5)=68
*  30 CHANGE N TO A$
*  40 PRINT A$
*  50 END
```

when executed, results in the printout of

```
%A6+D
```

indicating that the operation has converted the five numeric values in locations 1 through 5 of the list into the corresponding ASCII characters, and concatenated them into the string A$.

Note that in this case, the string must be dimensioned by the N(0)=X statement; otherwise, an error message INVALID CHANGE IN XX results, and the program halts. If the string is dimensioned too small, the string variable is truncated. If it is dimensioned too large, the string may contain irrelevant characters.

Strings may be of any length from 1 to 132 characters.

The following sample program illustrates another use of the CHANGE statement.

```
*  10 DIM A(100)
*  20 FOR I = 1 TO 26
*  30 LET A(I) = 64 + I
*  40 NEXT I
*  45 REM AT THIS POINT THE A LIST IS 65,66,67...90
*  50 LET A(0)=20
*  60 REM CONVERT ONLY THE 1ST 20 CODES IN A
*  70 REM      TO EQUIVALENT CHARACTERS
*  80 CHANGE A TO B$
*  90 PRINT B$
* 100 END
* RUN

   ABCDEFGHIJKLMNOPQRST
```

Statement 80 causes the conversion of numerics to their equivalent string characters. Statement 50 provides a count of the number of characters the user wishes to convert.

Table 9-1 lists the string characters and their equivalent numeric code.

Table 9-1. Numeric Code Table

String Characters	Code No. (decimal)	String Characters	Code No. (decimal)
(blank)	32	@	64
	33	A	65(97)
"	34	B	66(98)
#	35	C	67(99)
$	36	D	68(100)
%	37	E	69(101)
	38	F	70(102)
'	39	G	71(103)
(	40	H	72(104)
)	41	I	73(105)
*	42	J	74(106)
+	43	K	75(107)
,	44	L	76(108)
-	45	M	77(109)
.	46	N	78(110)
/	47	O	79(111)
0	48	P	80(112)
1	49	Q	81(113)
2	50	R	82(114)
3	51	S	83(115)
4	52	T	84(116)
5	53	U	85(117)
6	54	V	86(118)
7	55	W	87(119)
8	56	X	88(120)
9	57	Y	89(121)
:	58	Z	90(122)
;	59	[	91
<	60	\	92
=	61	]	93
>	62	↑	94
?	63		

Numerics in parentheses indicate lower case

Additional symbols useful on output are:

←(backward arrow)95	LF (line feed)10
EOT (end of transmission)4	CR (carriage return)13
BELL (rings bell in Teletype)7	RUB-OUT (tape use only)127

NOTES: 1. This is not a complete list - there are 128 characters numbered 0 through 127. Some of these numbers duplicate the above (on some teletypes) and some are just spaces.

2. The EOT character hangs up the phone if it is sent to a Model 33 Teletype.

o READ and DATA

READ and DATA statements are utilized in the conventional manner to manipulate alphanumeric data. A READ statement can be a mix of both numeric variables or string variables or can simply contain string variables. In turn, the DATA statement lists the sequence of data to correspond to the variables listed in the READ statement. Strings in a DATA statement must be enclosed in quotation marks if they begin with a digit or have an embedded comma. For example,

```
* 10 READ A,B$,C,D$,E$,F
        .
        .
* 90 DATA 85,XYZ,5,"4FG","MAY 26,1969",20
```

A leading blank in a string listed in the DATA statement is ignored unless the blank and its string are enclosed in quotes.

o PRINT

Strings are printed in the conventional manner; i.e., all forms of the PRINT statement are applicable when alphanumeric data is to be printed. For example,

```
* 10 READ A$,B$,C$
* 20 PRINT C$;B$;A$
* 30 DATA ING,SHAR,TIME-
* 40 END
* RUN
```

results in the printout of

```
TIME-SHARING
```

o INPUT

The requirements for handling alphanumeric data in an INPUT statement correspond to those of the READ statement in that the INPUT statement can be a mix of both numeric and string variables or can contain only string variables. For example,

```
* 10 INPUT X,Y$,Z
```

If the string variable represents a string with an embedded comma, the string, when entered during program execution, must be enclosed in quotes. A leading blank in a string is ignored unless the blank and its string are enclosed in quotes.

o RESTORE

Numeric data and string data are stored independently within two separate blocks of the BASIC system. The conventional RESTORE statement restores both numeric and string data. If the user wishes to restore only numeric data, he must use RESTORE followed by an asterisk:

```
* 10 RESTORE*
```

If the user wishes to restore only string data, he must use RESTORE followed by the $ character:

```
* 10 RESTORE$
```

Additional functions pertaining to string manipulation are available. These functions are CLK$ (to provide time of day) DAT$ (to provide calendar date), SST(X$,Y,Z)(to extract selected characters of a string), and LEN(X$)(to determine the number of characters in a specified string variable). Refer to "Additional Functions," in this section, for details concerning use of these functions.

ASCII DATA FILES

BASIC provides the means for creating files of data to be read, written on, or otherwise manipulated, all within the confines of the BASIC subsystem. A data file to be used as input must be prepared in advance and must be saved before it can be used in a program. A data file on which output is to be written during execution of a program does not necessarily need to have been created before that program is executed. If not in the user's catalog of permanent files when needed for output, a file is created as temporary, and can be changed to permanent status at logoff time. (Refer to "Saving of Temporary Files" in this section.) Data files can be created with or without line numbers. Data in a data file may range from zero to an unlimited number of characters.

All files are initially in read mode. A file can be placed in write mode by the use of a SCRATCH # statement. Read mode can be re-established by use of the RESTORE # statement.

Data files are implemented by data file input/output statements that supplement BASIC language statements. These data file input/output statements can be categorized as follows:

o File preparation statements

```
FILES filename 1, password;....;filename n, password
FILES user-id/catalogname$password/.../
      filename$password,permissions
FILE # file designator, "filename, password".
```

o File read statements

```
READ # file designator, input list
INPUT # file designator, input list
```

o File write statements

```
WRITE # file designator, output list
PRINT # file designator, output list
PRINT # file designator, USING statement number, output list
```

o Matrix input statements

```
MAT READ # file designator, matrix input list
MAT INPUT # file designator, matrix input list
```

o Matrix output statements

```
MAT WRITE # file designator, matrix output list
MAT PRINT # file designator, matrix output list
```

o File manipulation statements

```
SCRATCH # file designator
RESTORE # file designator
BACKSPACE # file designator
```

o Utility statements

```
APPEND # file designator
MARGIN # file designator, expression
DELIMIT #  file designator,  (character)
                             (abbreviation)

IF END # file designator,    THEN  line number
                             GOTO

IF MORE # file designator,   THEN  line number
                             GOTO
```

ASCII DATA FILE INPUT/OUTPUT STATEMENT FORMATS

The formats of data file input/output statements are described below. All statements, excepting FILES (used for initial data file preparation), make use of a data "file designator," a numeric argument whose value is used to select the data file desired for current operation. The numeric argument may be an integer, variable (subscripted or unsubscripted) or an arithmetic expression. The file designator is always preceded by a number sign (#).

File Preparation Statements

FILES

Purpose: To establish a relationship between numeric file designators and alphanumeric file names.

Format 1: FILES <filename 1,password;...;filename n,password>

Format 2: FILES <user-id/catalogname$password/.../filename$password, permissions>

Examples:
```
*10 FILES MONDAY;TUESDAY,PASS1
*10 FILES USERA/CAT1$PC/FIL1$PF1,R,W
```

Rules:

1. Semicolons are used as filename separators.
2. Filename passwords (if any) are separated from filenames by commas in Format 1 and by commas or dollar signs in Format 2. Where the slant (/) does not precede a password, a comma can be used.
3. An asterisk can be used in place of a filename, in which case the filename can be filled in via a FILE # statement (described below).
4. The filename of a data file must be referenced in a FILES statement before its first use within a program.
5. Multiple FILES statements are permissible within one program; one program is limited to eight named files.
6. Filenames cannot be duplicated within a set of FILES statements for a given program.
7. For Format 2, there is a 3-level limitation of catalog structure on files to be accessed. To exceed this 3-level limitation, the ACCESS subsystem must be used. See "File Access" in this section.

Remarks: The FILES statement sets all named data files to read mode.

Format 1 limits the user to the ability of accessing files contained in the user's master catalog. Format 2 permits the user to access files emanating from the user's subcatalogs or from catalogs and subcatalogs belonging to another user. The user, of course, must know the other user's identification, catalog and file names, and any required passwords. General or specific permissions for files are established by the files originator. Legal permission combinations are:

READ
WRITE
APPEND
READ,WRITE
READ,APPEND

Additional examples of the use of Format 2 may prove helpful.

```
*10 FILES USER1/CAT1$PC1/CAT2/CAT3/FIL1$PF1,R,W
```

Three levels of catalog structure (the limit) are accessed to get to FIL1, another user's file. Read and write permissions for the file are requested.

```
*10 FILES FIL2;USERB/FIL3,R,W;FIL4,PW4
```

Three files are accessed. FIL2 and FIL4 are the user's own files. FIL3 is a file originated by a user identified as USERB. Read and write permissions are being requested for FIL3.

```
*10  FILES/CATU/FIL7;USERD/CATD$PW/FIL8,R,W
```

Two files are accessed. FIL7 is the user's own file located in catalog CATU. FIL8 is a file originated by user USERD. Read and write permissions are being requested for FIL8.

FILE

Purpose: To permit replacement of a data file, or to permit specification of a data file indicated by an asterisk in a FILES statement.

Format: FILE # <file designator, "filename,password">

Examples:

1.
```
*10 FILES A;B;C
     . .
     . .
     . .
*50 FILE #3 ,"D"
```

Data file C, the third file, is replaced by data file D.

2.
```
*10 FILES A;*;C
     . .
     . .
     . .
*50 FILE #2 ,"B"
```

The asterisk-indicated data file, the second file, is specified as data file B.

Rules:

1. The filename can be indicated as follows:

   a. filename and password (if any) enclosed in quotes

   b. string variables (subscripted or unsubscripted) for filename and password (if any)

   c. asterisk enclosed in quotes (see remarks below)

2. A file named in a FILE # statement cannot appear in a FILES statement, unless the file has been released before its use in the FILE # statement.

3. One program is limited to eight named files.

Remarks: When a quote-enclosed asterisk is used as a "filename," the associated file designator is invalidated until such time that it is validated again by a subsequent FILE statement. For example:

```
*10 FILES A;B;C
       . .
       . .
       . .
*50 FILE #3 ,"*"
```

In statement 50, file designator 3 now refers to a null filename and cannot be used again until it is reset by another FILE # statement.

A colon (instead of a comma) can be used as the separator between file designator and "filename."

A string variable can be substituted for "filename" if the string variable contains the filename to be referenced. For example:

```
*10 FILES MONDAY;TUESDAY
*20 LET A1$ = "SATURDAY"
*30 FILE #1,A1$
```

File Read Statements

READ

Purpose: To read data from a data file into an input list.

Format: READ # <file designator, input list>

Example:

```
*10 FILES MONDAY;TUESDAY
*20 READ #1,X1,A1$,X2,A2$
```

If data file MONDAY is represented by

```
10 5.6, SEPTEMBER, 100.5, OCTOBER
```

at execution time, the real value of 5.6 would be read into X1, string SEPTEMBER into A1$, real value 100.5 into X2, and string OCTOBER into A2$.

Rules:

1. The input list must consist of delimiter-separated variables, numeric or string, any of which can be subscripted.

2. When an input list contains both numeric and string variables, data elements in the data file must correspond one-to-one to the input list.

3. If the file designator is zero, data is read from internal data created by the program's DATA statement(s). For reading of internal data, there need not be a one-to-one correspondence between numeric and string variables in the input list and data file.

4. A colon can be used in the READ # statement instead of a comma to separate file designator from the input list.

Remarks: The line number of a data file is not part of the data read by a file read statement into an input list. At least one blank should separate the line number from data in the data file.

If an entire data file is not read because of insufficient variables in the input list of a file read statement, the word pointer remains positioned after the last data item read until additional file read statement(s) are executed.

If the first character of an input string is a quote ("), the string must be terminated by a delimiter following the trailing quote. The resulting string consists of the characters enclosed by the quotes.

Data files to be read by the READ # statement require that elements of each data line be delimiter-separated. A delimiter may or may not end the line, the decision being left to the user.

Purpose: To read data from a data file into an input list, treating line numbers as data items.

Format: INPUT # <file designator, input list>

Example:

```
*10 FILES MONDAY,TUESDAY
*20 INPUT #1,A,B,C,D,E
```

If data file MONDAY is represented by

```
  10 1,2,3,4,5
```

the statement

```
*30 PRINT A;B;C;D;E
```

would produce

```
  101 2 3 4 5
```

at program execution time.

Rules:

1. The input list must consist of comma-separated variables, numeric or string, any of which can be subscripted.
2. When an input list contains both numeric and string variables, data elements in the data file must correspond one-to-one to the input list.
3. A colon may be used in the INPUT statement instead of a comma to separate the file designator from the input list.
4. If the file designator is zero, at execution time the program asks for data from the user's terminal. In response to a question mark, the user supplies data elements to correspond to the input list.

Remarks: Embedded blanks within a line number causes misinterpretation in reading of a line number.

If the first character of an input string is a quote ("), the string must be terminated by a specified delimiter following the trailing quote. The resulting string consists of the characters enclosed by the quotes.

File Write Statements

WRITE

Purpose: To generate a data file in which each line contains a line number and data elements delimiter-separated.

Format: WRITE # <file designator, output list>

Example:

```
*10 FILES SUNDAY; MONDAY; ABC
*20 READ #2, X1, A1$
*30 SCRATCH #3
*40 WRITE #3, X1, A1$
```

If data file MONDAY is represented by

```
10  5, OCTOBER, 1969
```

the WRITE # statement generates a new data file ABC with contents of

```
10  5, OCTOBER
```

Data file ABC can be a temporary or permanent file.

Rules:

1. The output list can consist of numeric or string variables (any of which can be subscripted), or arithmetic expressions.
2. The format conventions of the normal PRINT statement apply to the WRITE # statement.
3. If the file designator is zero, the generated data file is written out to the user's terminal upon program execution, with no SCRATCH # statement required.
4. A colon can be used in the WRITE statement instead of a comma to separate the file designator from the output list.
5. The standard line length is equal to 75 characters, including line numbers. The MARGIN statement can be used to adjust a line from 2 to 160 characters.

Remarks: The WRITE # statement generates a data file that begins with line number 10 and increments by 10 for each additional line. Each line number is separated from the first data element of the line by at least one blank. Data elements, in turn, are separated by delimiters (commas or user-specified delimiters).

When the TAB(X) function is used, the line number is included in the count for the tab position.

A data file generated by a WRITE # statement is equivalent to a data file saved in the conventional manner; i.e., the file can serve as input to other subsystems (e.g., LIST).

Purpose: To generate a data file that contains no line numbers or delimiters on printout.

Format: PRINT # <file designator, output list>

Example:

```
*10 FILES SUNDAY;MONDAY;ABC
*20 INPUT #2,X1,A1$
*30 SCRATCH #3
*40 PRINT #3,X1,A1$
```

If data file MONDAY is represented by

```
5,OCTOBER,1969
```

the PRINT # statement generates a new data file ABC with contents of

```
5   OCTOBER
```

Rules:

1. The output list can consist of numeric or string variables (any of which can be subscripted), arithmetic expressions, or string constants (literals) in quotes.

2. The format conventions of the normal PRINT statement apply to the PRINT # statement.

3. If the file designator is zero, the generated data file is printed out on the user's terminal upon program execution, with no SCRATCH # statement required.

4. A colon can be used in the PRINT # statement instead of a comma to separate the file designator from the output list.

5. The standard line length is equal to 75 characters including line numbers. The MARGIN statement can be used to adjust a line from 2 to 160 characters.

6. No delimiters are created by the PRINT # statement.

Remarks: The PRINT # and WRITE # statements are utilized in similar fashions. The difference lies in the manner in which the generated data file is printed out. With the use of the PRINT statement, no line numbers or data element delimiters (commas or semicolons) appear.

A data file generated by a PRINT # statement can serve as input to other subsystems (e.g., LIST).

Purpose: To provide the ability to format data written to a data file.

Format: PRINT # <file designator> ,<USING statement number, output list>

Where:

"statement number" is number of a statement in the program that contains format control characters and printable constants; "output list" consists of comma-separated arguments to be printed in sequential order.

Example:

```
*10  FILES FORMAT
*20  SCRATCH #1
*30  A = 123.45
*40  B = -3.456
*50  C = -.017
*60  PRINT #1,USING 80,A,B,C
*70  PRINT #1,USING 90,A,B,C
*80:DECIMAL  FIELDS   ###.##  ##.###  #.###
*90:EXPONENT FIELDS   ##.###↑↑↑↑  ##.###↑↑↑↑  ##.###↑↑↑↑
*100 END

* RUN
* LIST FORMAT

DECIMAL  FIELDS      123.45       -3.456      -.017
EXPONENT FIELDS      12.345E 01   -3.456E 00  -1.700E-02
```

Rules:

1. The statement number named in a PRINT # USING statement points to an "image" statement which formats the line to be printed. The image statement is of the form

   statement number: image

2. The image of an image statement (colon-separated from the statement number) consists of format control characters and printable constants.

3. Format control characters are as follows:

   ' (apostrophe) - a 1-character field that is filled with the first character in an alphanumeric string, regardless of string length.

   # (number sign) - the replacement field for a numeric character; each # specifies a space for one digit.

   ↑↑↑↑ (four up-arrows) - specifies scientific notation for a numeric field (E-format).

4. Printable constants are all characters other than format control characters.

Remarks: The image of an image statement can consist of one or more of the following fields:

- integer
- decimal
- exponential
- alphanumeric
- literal

Refer to "Formatting Line Output" in this section for details concerning use of format control statement.

Data to be retrieved from a data file via READ # or INPUT # statements should not be placed on the file by a PRINT # USING statement. Data files containing data formatted by PRINT # USING statements are intended for terminal printout only by the way of the LIST command.

Matrix Input Statements

MAT READ

Purpose: To read data from data file into a matrix input list.

Format: MAT READ # <file designator, matrix input list>

Example:

```
*10 FILES A;B
*20 DIM M1(3,3),M2(5,7)
*30 MAT READ #1,M1,M2
```

If data file A is represented by

```
10 1,2,3,.....,10,
 .    .
 .    .
 .    .
50 ........ 48,49,50,
```

M1 contains the matrix

```
 1  2  3
 4  5  6
 7  8  9
```

M2 contains the matrix

```
 10  11  12  13  14  15  16
 17  18  19  20  21  22  23
 24  25  26  27  28  29  30
 31  32  33  34  35  36  37
 38  39  40  41  42  43  44
```

Rules:

1. String variables cannot be used in the matrix input list.
2. Matrices in the matrix input list can have their dimensions specified in a DIM statement or in the MAT READ # statement itself.
3. When a matrix in the matrix input list is not dimensioned, a 10 by 10 matrix is assumed.
4. Files to be read by a MAT READ # statement must contain line numbers.
5. A colon can be used in the MAT READ # statement instead of a comma to separate the file designator from the matrix input list.

Remarks: If the file designator is zero, internal data is to be read from user-supplied DATA statement(s) within the program.

If there are not enough data elements in a data file to fill a designated matrix, the matrix is filled out with zeros.

Purpose: To read data from a data file into a matrix input list, treating line numbers as data items.

Format: MAT INPUT # <file designator, matrix input list>

Example:

```
*10 FILES M1
*20 DIM M2(3,3)
*30 MAT INPUT #1,M2
```

If data file M1 contains

```
10  1,2,3,4,5,6,7,8,9
```

M2 contains the matrix

```
101   2   3
  4   5   6
  7   8   9
```

Rules:

1. String variables cannot be used in the matrix input list.
2. Matrices in the matrix input list can have their dimensions specified in a DIM statement or in the MAT INPUT # statement itself.
3. When a matrix in the matrix input list is not dimensioned, a 10 x 10 matrix is assumed.
4. A colon can be used in the MAT INPUT # statement instead of a comma to separate the file designator from the matrix input list.

Remarks: If the file designator is zero, at execution time the program asks for data from the user's terminal. In response to a question mark, the user supplies data elements to correspond to the input list.

If there are not enough data elements in a data file to fill a designated matrix, the matrix is filled out with zeros.

Matrix Output Statements

MAT WRITE

Purpose: To write matrices specified in a matrix output list to designated data file(s).

Format: MAT WRITE # <file designator, matrix output list>

Example:

```
*10 FILES A;B;C
*20 DIM M1(3,3),M2(5,7)
*30 MAT READ #1,M1,M2
*40 SCRATCH #2
*50 MAT WRITE #2,M1,M2
```

Matrices M1 and M2, read from data file A, are written to data file B.

Rules:

1. String variables cannot be used in the matrix output list.
2. Matrices in the matrix output list must have their dimensions specified in a DIM statement; they cannot be dimensioned in a MAT WRITE # statement.
3. A colon can be used in the MAT WRITE statement instead of a comma to separate the file designator from the matrix output list.

Remarks: The MAT WRITE # statement generates a data file that begins with line number 10 and increments by 10 for each additional line. Each line number is separated from the first data element of the line by a blank.

Purpose: To write matrices specified in a matrix output list to a designated data file that contains no line numbers or delimiters on printout.

Format: MAT PRINT # <file designator, matrix output list>

Example:

```
*10 FILES M1,M2
*20 MAT INPUT #1,A(2,3)
*30 SCRATCH #2
*40 MAT PRINT #2,A
```

If data file M1 is represented by

```
1,2,3,4,5,6
```

The MAT PRINT # statement generates a new data file M2 which consists of

```
1    2    3
4    5    6
```

Rules:

1. String variables cannot be used in the matrix output list.
2. Matrices in the matrix output list must have their dimensions specified in a DIM statement; they cannot be dimensioned in a MAT PRINT # statement.
3. A colon can be used in the MAT PRINT # statement instead of a comma to separate the file designator from the matrix output list.

Remarks: The MAT PRINT # and MAT WRITE # statements are utilized in similar fashions. With the use of the MAT PRINT # statement, no line numbers or data element delimiters appear.

A data file generated by a MAT PRINT # statement can serve as input to other subsystems (e.g., LIST).

If the file designator is zero, the generated data file is printed out at the user's terminal upon program execution.

File Manipulation Statements

SCRATCH

Purpose: To place a data file in write mode.

Format: SCRATCH # <file designator>

Example:

```
*10 FILES DEBITS;CREDITS
*20 READ #1,X1,X2,X3
*30 SCRATCH #2
*40 WRITE #2,X1,X2,X3
```

Data file CREDITS is placed in write mode by SCRATCH # statement 30, prior to being written on by WRITE # statement 40.

Remarks: A SCRATCH # statement deletes all data previously contained in the designated file providing the file has been written on; i.e., for files created by WRITE #, MAT WRITE #, or PRINT # statements.

If the data file CREDITS is a file not previously created and saved, the file system queries the user as to the disposition of the file.

RESTORE

Purpose: To position the data pointer for the designated data file to the beginning of the file and permit the file to be read.

Format: RESTORE # <file designator>

Examples:

1.

```
*10 FILES A;B;C
*20 READ #1,X1,X2,X3
*30 RESTORE #1
*40 READ #1,Y1,Y2,Y3
```

RESTORE # statement 30 permits data from data file A to be read again.

2.

```
*10 FILES A;B;C
*20 READ #1,X1,X2,X3
   .    .
   .    .
*50 SCRATCH #1
*60 WRITE #1,Y1,Y2,Y3
*70 RESTORE #1
*80 READ #1,X1,X2,X3
```

RESTORE # statement 70 places data file A in read mode and permits data just written to be read.

Remarks: If a designated data file is in write mode as the result of a SCRATCH # statement, a RESTORE # statement repositions the data pointer to the beginning of the file and places the file in read mode.

Purpose: To position the data pointer for the designated data file backward one delimiter.

Format: BACKSPACE # <file designator>

Example: If data file A contains

```
10  1,2,3,4,5,

20  6,7,8,9,10,
```

The Program

```
*10 FILES A;B;C
*20 READ #1,X1,X2,X3,X4,X5,X6,X7
*30 FOR I = 1 to 4
*40 BACKSPACE #1
*50 NEXT I
*60 READ #1,Y1,Y2,Y3,Y4
*70 PRINT X1,X2,X3,X4,X5,X6,X7
*80 PRINT Y1,Y2,Y3,Y4
*90 END
*RUN
```

produces

```
1  2  3  4  5  6  7

4  5  6  7
```

Remarks: The BACKSPACE # statement places the designated file in read mode.

If the designated file is backspaced past the beginning of the file, the data pointer is positioned to the beginning of the file.

Utility Statements

APPEND

Purpose: To permit data to be added to a designated file.

Format: APPEND # <file designator>

Example:

```
*10 FILES A;B;C
*20 READ #1,X1,X2,A1$
*30 APPEND #2
*40 WRITE #2,X1,X2,A1$
```

APPEND # statement 30 places data file B in write mode and permits WRITE # statement 40 to append data to data already on B.

Remarks: When the APPEND # statement is executed, the data pointer for the designated file is moved immediately past the last data item on the file. The file is also placed in write mode, ready to accept the next WRITE # statement.

MARGIN

Purpose: To permit the specification of the rightmost character position for a designated file.

Format: MARGIN # <file designator, expression>

Example:

```
*10 FILES A;B;C
*20 SCRATCH #1
*30 SCRATCH #2
*40 MARGIN #1,20
*50 MARGIN #2,M*N-5
*60 WRITE #1,X1,X2,X3,X4
*70 WRITE #2,Y1,Y2,Y3,Y4
```

Rules:

1. The standard line (record) length for files created by WRITE # or PRINT # statements is 75 characters, including the line number. By use of the MARGIN # statement, the user can explicitly specify a maximum line length for a designated file to be any value between 2 and 160 characters. If the specified line length exceeds the physical capability of the terminal in use, the result can be a character-overprint at the end of the line.

2. A colon can be used in the MARGIN # statement instead of a comma to separate the file designator from the expression.

3. A file designator of zero is interpreted as being the user's terminal.

Purpose: To permit the use of a delimiter other than a comma in a designated file.

Format: DELIMIT # file designator, $\left\{\begin{array}{l}\text{(character)}\\ \text{(abbreviation)}\end{array}\right\}$

Example:

```
*10 FILES INPUT;OUTPUT
*20 READ #1,A,B,C,D,E,F
*30 DELIMIT #2,(LF)
*40 SCRATCH #2
*50 WRITE #2,A;B;C;D;E;F
```

If data file INPUT contains

```
10  1,2,3,4,5,6
```

a printout of data file OUTPUT would produce

```
10  1
      2
         3
            4
               5
                  6
```

Rules:

1. The standard delimiter separating data elements in a data file is the comma. The DELIMIT # statement can specify any character, or abbreviation for non-printing character(s).

2. Non-printing character abbreviations (e.g.,CR for carriage return; LF for line feed) are those specified by ASCII. Refer to Appendix C for a list of octal/ASCII conversion equivalents.

3. A DELIMIT # statement must be used prior to its associated READ # or WRITE # statement.

4. A file designator of zero is interpreted as being the user's terminal and the DELIMIT # statement is ignored.

Remarks: A PRINT # statement results in the printout of designated data without delimiters (or line numbers) regardless of whether standard or nonstandard delimiters are used.

IF END #----THEN
or
IF END #----GOTO

Purpose: To provide for a means of testing for the end of data when reading a data file.

Format: IF END # <file designator> {THEN / GOTO} <statement number>

Example:

```
*10 FILES A;B
*20 READ #1,X1,X2,X3
*30 PRINT X1,X2,X3,
*40 IF END #1 THEN 60
*50 GOTO 20
*60 PRINT "OUT OF DATA IN FILE A"
*70 END
*RUN
```

If data file A contains

```
10  1,2,3,4,5,6,7,
20  8,9,10,
```

the executed program produces

```
1  2  3  4  5  6  7  8  9  10  0  0
OUT OF DATA IN FILE A
```

Rules: A comma or a colon can be used in an IF END #---THEN statement to separate the file designator from the THEN portion of the statement.

Remarks: If data elements (or string data) of a data file are exhausted before the input list in a READ # or MAT READ # statement is satisfied, the list is filled out by zeros (or null) upon program execution.

The IF END #---THEN statement directs the system to go to a designated out-of-sequence statement when no more data remains on the file.

IF MORE #---THEN
or
IF MORE #---GOTO

Purpose: To provide for a means of testing to determine whether at least one valid data element remains on a data file when reading the file.

Format: IF MORE # <file designator> $\begin{Bmatrix} \text{THEN} \\ \text{GOTO} \end{Bmatrix}$ <statement number>

Example:

```
*10 FILES A;B
*20 READ #1,X1,X2,X3
*30 PRINT X1,X2,X3,
*40 IF MORE #1 THEN 20
*50 PRINT "OUT OF DATA IN FILE A"
*60 END
*RUN
```

If data file A contains

```
10  1,2,3,4,5,6,7,

20  8,9,10,
```

the executed program produces

```
1  2  3  4  5  6  7  8  9  10  0  0

OUT OF DATA IN FILE A
```

Rule: A comma or a colon can be used in an IF MORE # ---THEN statement to separate the file designator from the THEN portion of the statement.

Remarks: If data elements (or string data) of a data file are exhausted before the input list of a READ # or MAT READ # statement is satisfied, the list is filled out by zeros (or null) upon program execution.

The IF MORE #---THEN statement directs the system to go to a designated out-of-sequence statement when more data remains on the file.

BINARY FILES

BASIC permits the user to perform file input/output with files made up in binary format. This mode of operation presupposes a sophisticated user whose knowledge encompasses the makeup of binary-type files and who has the need to create programs that have special applications.

The use of binary input/output, as contrasted with the use of alphanumeric (ASCII) input/output, speeds up program execution and compacts file space. However, data cannot be placed on a binary file directly from the user's terminal, nor can a binary file be listed (by means of the LIST command) so as to verify its content.

Binary files can be either sequential or random and can be written, read, backspaced, scratched, and restored. Data can be appended to the end of a sequential binary file. Any word on a random binary file is accessible for reading or writing without the need for traversing the file space which precedes the word. When a random binary file is to be created, file space must be obtained by means of the ACCESS subsystem (see "File Access" below).

A word pointer is maintained in the file control block of each binary file so as to indicate the next word of the file to be read or written. Each binary file consists of a number of words, zero through n-1. For sequential files, the word pointer is initially set to word zero and moved forward with each READ: and WRITE: statement. The word pointer can be moved backward by means of the RESTORE:, SCRATCH:, and BACKSPACE: statements. This same forward and backward movement of the word pointer through statement manipulation exists for random files, with the exception that the user can alter the position of the word pointer by means of an additional statement--SET:. If the user wishes to begin reading and writing of a random file at a position other than word zero, he can position the word pointer to any position within the file with the SET: statement and begin his reading or writing at that point. The current position of the word pointer for a random file and the current length of a random file can be determined by use of functions LOC and LOF.

Each numeric data element on a binary file occupies one word and is in single-precision, floating-point format. Alphanumeric strings, that can vary in length from 1 to 132 characters, are placed on binary files with a string control word on either end of the string. Each string thus occupies two words for control, plus enough words to contain the actual string of characters at four characters per word. The user must exercise caution in manipulating the word pointer on random binary files containing strings. A SET: statement could inadvertently position the word pointer to the middle of a string, causing an error in the next read or write. The user must take care to position the word pointer to a leading string control word and see to it that extended strings do not destroy data already on a file.

All sequential files are initially in read mode. A file can be placed in write mode by the use of SCRATCH: statement. Read mode can be re-established by the use of the RESTORE: statement. Read/write mode does not apply to random files, which can be read or written at any point at any time.

Binary files are implemented by binary file input/output statements that supplement BASIC language statements. These binary file input/output statements are categorized as follows and, unless indicated, apply to both sequential and random binary files:

- File preparation statements

  FILES filename 1,password;....;filename n,password

  FILES user-id/catalogname$password/.../
  filename$password,permissions

  FILE: file designator, "filename,password"

- File read statement

  READ: file designator,input list

- File write statement

  WRITE: file designator,output list

- Matrix input statement

  MAT READ: file designator,matrix input list

- Matrix output statement

  MAT WRITE: file designator, matrix output list

- File manipulation statements

  SCRATCH: file designator

  RESTORE: file designator

  BACKSPACE: file designator

- Utility statements

  APPEND: file designator
  (for sequential files only)

  If END: file designator $\left\{\begin{matrix}\text{THEN}\\\text{GOTO}\end{matrix}\right\}$ line number

  IF MORE: file designator $\left\{\begin{matrix}\text{THEN}\\\text{GOTO}\end{matrix}\right\}$ line number

  SET: file designator TO expression
  (for random files only)

The current position of the word pointer for a random binary file or its current length can be determined by the use of special functions. These functions are as follows:

- Word pointer location

  LOC(file designator)

- File length

  LOF(file designator)

Upon program execution, these functions contained within a program cause the printout of integers, indicating the desired word numbers.

NOTE: For all practical purposes, the IF END: and IF MORE: statements are applicable to sequential files only. Random files have no logical end-of-data; the entire random file supposedly contains good data and is accessible at any point for reading and writing. Thus, if a random file has a current size of three blocks (960 words) and has data written in only he first 10 words, the IF END: and IF MORE: statements cannot be used to determine when the end of the first 10 words has been reached. The remaining 950 words are accessible data despite the fact that they are empty.

BINARY FILE INPUT/OUTPUT STATEMENT FORMATS

The formats of binary file input/output statements are described below. All statements, excepting FILES (used for initial binary file preparation) make use of a "file designator," a numeric argument whose value is used to select the binary file desired for current operation. The numeric argument can be an integer, a variable (subscripted or unsubscripted), or an arithmetic expression. The file designator is always preceded by a colon.

File Preparation Statements

FILES

Purpose: To establish a relationship between numeric file designators and alphanumeric file names.

Format 1: FILES <filename 1,password;...;filename n,password>

Format 2: FILES <user-id/catalogname$password/.../>
filename$password,permissions

Examples: *10 FILES MONDAY;TUESDAY,PASS1

*10 FILES USERA/CAT1$PC/FIL1$PF1,R,W

Rules:

1. Semicolons are used as filename separators.

2. Filename passwords (if any) are separated from filenames by commas in Format 1 and by commas or dollar signs in Format 2. Where the slant (/) does not precede a password, a comma can be used.

3. An asterisk can be used in place of a filename, in which case the filename can be filled in via a FILE: statement (described below).

4. The filename must be referenced in a FILES statement before its first use within a program.

5. Multiple FILES statements are permissible within one program; one program is limited to eight named files.

6. Filenames cannot be duplicated within a set of FILES statements for a given program.

7. For Format 2, there is a 3-level limitation of catalog structure on files to be accessed. To exceed this 3-level limitation, the ACCESS subsystem must be used. See "File Access" in this section.

Remarks: The FILES statement sets all named sequential binary and ASCII files to read mode.

Format 1 limits the user to the ability of accessing files contained in the user's master catalog. Format 2 permits the user to access files emanating from the user's subcatalogs or from catalogs and subcatalogs belonging to another user. The user, of course, must know the other user's identification, catalog and file names, and any required passwords. General or specific permissions for files are established by the files' originator. Legal permission combinations are:

READ
WRITE
APPEND
READ,WRITE
READ, APPEND

Additional examples of the use of Format 2 may prove helpful.

```
*10 FILES USER1/CAT1$PC1/CAT2/CAT3/FIL1$PF1,R,W
```

Three levels of catalog structure (the limit) are accessed to get to FIL1, another user's file. Read and write permissions for the file are requested.

```
*10 FILES FIL2;USERB/FIL3,R,W;FIL4,PW4
```

Three files are accessed. FIL2 and FIL4 are the user's own file. FIL3 is a file originated by a user identified as USERB. Read and write permissions are being requested for FIL3.

```
*10 FILES/CATU/FIL7;USERD/CATD$PW/FIL8,R,W
```

Two files are accessed. FIL7 is the user's own file located in his catalog CATU. FIL8 is a file originated by user USERD. Read and write permissions are being requested for FIL8.

FILE:

Purpose: To permit replacement of a binary file by another binary filename, or to permit specification of a binary file indicated by an asterisk in a FILES statement.

Format: FILE: <file designator, "filename,password">

Examples: 1.

```
*10 FILES A;B;C
      . .
      . .
      . .
*50 FILE: 3,"D"
```

Binary file C, the third file, is replaced by binary file D.

2.

```
*10 FILES A;*;C
      . .
      . .
      . .
*50 FILE: 2,"B"
```

The asterisk-indicated binary file, the second file, is specified as binary file B.

Rules:

1. The filename can be indicated as follows:
   a. filename and password (if any) enclosed in quotes
   b. string variables (subscripted or unsubscripted) for filename and password (if any)
   c. asterisk enclosed in quotes (see Remarks below)
2. A file named in a FILE: statement cannot appear in a FILES statement, unless the file has been released before its use in the FILE: statement.
3. One program is limited to eight named files.

Remarks:

When a quote-enclosed asterisk is used as a "filename," the associated file designator is invalidated until such time that it is validated again by a subsequent FILE statement. For example:

```
*10 FILES A;B;C
      . .
      . .
      . .
*50 FILE: 3,"*"
```

In statement 50, file designator 3 now refers to a null filename and cannot be used again until it is reset by another FILE: statement.

A colon (instead of a comma) can be used as the separator between file designator and "filename."

A string variable can be substituted for "filename" if the string variable contains the filename to be referenced. For example:

```
*10 FILES MONDAY;TUESDAY
*20 LET A1$ = "SATURDAY"
*30 FILE: 1,A1$
```

File Read Statement

READ:

Purpose: To read binary data from a permanent binary file into an input list.

Format: READ: <file designator, input list>

Example: The binary file SUNS contains a list of the names of basketball players, with each player's score average following his name. The beginning of the file (the first three names) could appear as follows:

Data	Word	Octal Representation
Control word	0	001600000700
HAWK	1	110101127113
INS	2	111116123040
Control word	3	001400000700
30	4	012740000000
Control word	5	001600000400
WALK	6	127101114113
Control word	7	001400000400
20	8	012500000000
Control word	9	001600001000
GOOD	10	107117117104
RICH	11	122111103110
Control word	12	001400001000
25	13	012620000000

The following program produces the names of the first three players and their score averages.

```
*10 FILES SUNS
*20 FOR I = 1 to 3
*30 READ:1,N$,S
*40 PRINT USING 60,N$,S
*50 NEXT I
*60:'LLLLLLLLLL    #
*70 PRINT
*80 PRINT "MORE TO COME"
*90 END
*RUN

HAWKINS         30
WALK            20
GOODRICH        25

MORE TO COME
```

Rules:

1. The input list must consist of delimiter-separated variables, numeric or string, any of which can be subscripted.

2. When an input list contains both numeric and string variables, data elements in the binary file must correspond one-to-one to the input list.

3. A colon can be used, instead of a comma, to separate the file designator from the input list.

Remarks: If an entire binary file is not read because of insufficient variables in the input list file read statement, the word pointer remains positioned after the last data item read until additional file read statement(s) are executed.

File Write Statement

WRITE:

Purpose: To write binary data on a permanent binary file.

Format: WRITE: <file designator,output list>

Example:

```
*10 FILES PHX1
*20 H1 = H2 = 5
*30 H3 = 6 H4 = 6.2
*40 S1$="BINARY"
*50 S2$="DATA"
*60 SCRATCH:1
*70 WRITE:1,H1,H2,H3,H4,S1$,S2$
*80 END
```

Upon program execution, the following data would be placed in binary file PHX1.

Data	Word	Octal Representation
5	0	006500000000
5	1	006500000000
6	2	006600000000
6.2	3	006614631463
Control word	4	001600000600
BINA	5	102111116101
RY	6	122131040040
Control word	7	001400000600
Control word	8	001600000400
DATA	9	104101124101
Control word	10	001400000400

The file's word pointer would be at word 11 of the file.

Rules:

1. The output list can consist of numeric or string variables (any of which can be subscripted), or arithmetic expressions.
2. The format conventions of the normal PRINT statement apply to the WRITE: statement.
3. A colon can be used in the WRITE: statement instead of a comma to separate the file designator from the output list.

Remarks: The word pointer for the referenced binary file is incremented by one after each word is written on the file.

Matrix Input Statement

MAT READ:

Purpose: To read data from permanent binary file into a matrix input list.

Format: MAT READ: <file designator,matrix input list>

Example: Assume that binary file INTEGERS contains the numbers 0 through 10 in its first 11 words. The following program can be used to read data from file INTEGERS into a matrix input list.

```
*10 FILES*;INTEGERS
*20 DIM M8(6)
*30 READ:2,N1,N2
*40 MAT READ:2,M8
*50 MAT PRINT M8
*60 END
```

Upon execution, the program would produce the following printout:

```
2

3

4

5

6

7
```

Rules:

1. String variables cannot be used in the matrix input list.

2. Matrices in the matrix input list must have their dimensions specified in a DIM statement or in the MAT READ: statement itself.

3. When a matrix in a matrix input list is not dimensioned, a 10 by 10 matrix is assumed.

4. A colon can be used, instead of a comma, to separate the file designator from the matrix input list.

Remarks: If there are not enough data elements in a binary file to fill a designated matrix, the matrix is filled out with zeros.

Matrix Output Statement

MAT WRITE:

Purpose: To write matrices specified in a matrix output list to designated permanent binary file.

Format: MAT WRITE: <file designator,matrix output list>

Example: Assume that binary file ABCD has been created via ACCESS as a random file. The following program can be used to write a matrix output list to file ABCD.

```
*10 FILES ABCD
*20 DIM T(2,3)
*30 T(1,1)=1 T(1,2)=2 T(1,3)=3
*40 T(2,1)=4 T(2,2)=5 T(2,3)=6
*50 SCRATCH:1
*60 SET:1 TO 4
*70 MAT WRITE:1,T
*80 END
```

Statement 60 could not be used if ABCD was not random.

Upon execution, file ABCD contains matrix T as follows:

Data	Word	Octal Representation
	0	400000000000
	1	400000000000
	2	400000000000
	3	400000000000
1	4	002400000000
2	5	004400000000
3	6	004600000000
4	7	006400000000
5	8	006500000000
6	9	006600000000

Rules:

1. String variables cannot be used in the matrix output list.
2. Matrices in the matrix output list must have their dimensions specified in a DIM statement; they cannot be dimensioned in a MAT WRITE: statement.
3. When a matrix in the matrix output list is not dimensioned, a 10 by 10 matrix is assumed.
4. A colon can be used, instead of a comma, to separate the file designator from the matrix output list.

File Manipulation Statements

SCRATCH:

Purpose: To place a binary file in write mode.

Format: SCRATCH: <file designator>

Example:

```
*10 FILES ABC;XYZ
*20 READ:1,X1,X2,X3
*30 SCRATCH:1
*40 WRITE:1,X1,X2,X3
```

Binary file ABC is placed in write mode by SCRATCH: statement 30, prior to being written on by WRITE: statement 40.

Remarks: A SCRATCH: statement deletes all data previously contained in the designated file; i.e., data written by WRITE: or MAT WRITE: statements.

The SCRATCH: statement can be used with both sequential and random binary files. For sequential files, the word printer is set to zero and the file is placed in write mode. For random files, the entire file is filled with floating point zeros and the word pointer is set to zero. The read/write mode does not apply to random file; therefore, the SCRATCH: statement need not be utilized with a random file unless the user wishes to clear the entire random file to zeros.

RESTORE:

Purpose: To position the word pointer for the designated binary file to the beginning of the file and permit the file to be read.

Format: RESTORE: <file designator>

Example:

```
*10 FILES HUGO
*20 R1=8.8
*30 R2=9.9
*40 R3=10.10
*50 R1$="THIS LINE SHOULD APPEAR TWICE"
*60 SCRATCH:1
*70 WRITE:1,R1,R2,R3,R1$
*80 RESTORE:1
*90 READ:1,S1,S2,S3,S1$
*100 PRINT R1$;R1;R2;R3
*110 PRINT S1$;S1;S2;S3
*120 END
*RUN
```

produces the printout

```
THIS LINE SHOULD APPEAR TWICE  8.8  9.9  10.1
THIS LINE SHOULD APPEAR TWICE  8.8  9.9  10.1
```

RESTORE: (statement 80) places binary file HUGO in read mode and permits data just written to be read.

Remarks: If a designated binary file is in write mode as a result of a SCRATCH: statement, a RESTORE: statement repositions the word pointer to the beginning of the file. The file is placed in read mode if it is sequential.

BACKSPACE:

Purpose: To position the word pointer for the designated binary file backward one data element.

Format: BACKSPACE: <file designator>

Example:

```
*10 FILES HIPPO
*20 A1=1 A2=2 A3=3
*30 E1$="IS A"
*40 E2$=" CROWD"
*50 SCRATCH:1
*60 WRITE:1,A1,A2,A3,E1$,E2$
*70 FOR J=1 TO 3
*80 BACKSPACE:1
*90 NEXT J
*100 READ:1,B3,G1$,G2$
*110 PRINT B3;G1$;G2$
*120 END
*RUN
```

produces the printout

```
3 IS A CROWD
```

Remarks: The BACKSPACE: statement places the designated binary file in read mode if the file is sequential. If the designated binary file is backspaced past the beginning of the file, the word pointer is positioned to the beginning of the file.

Utility Statements

APPEND:

Purpose: To permit data to be added to a designated, sequential binary file.

Format: APPEND: <file designator>

Example: Assume that the binary file SEE is a sequential file containing the integers 1 through 15.

```
*10 FILES A;B;SEE
*20 APPEND:3
*30 FOR I=16 TO 20
*40 WRITE:3,I
*50 NEXT I
*60 END
*RUN
```

The executed program appends the integers 16 through 20 to the file SEE.

Rules: The APPEND: statement applies to sequential files only.

Remarks: The APPEND: statement sets the word pointer for the designated file to the position immediately following the last data word. The file is then placed in write mode, ready to accept the next WRITE: statement.

IF END:----THEN
or
IF END:----GOTO

Purpose: To provide a means of testing for end of data when reading a binary file.

Format: IF END: <file designator> {THEN / GOTO} <statement number>

Example:

```
*10 FILES ZORRO
*20 K1=1
*30 A$="EACH STRING "
*40 B$="HAS A "
*50 C$="LEADING AND TRAILING "
*60 D$="CONTROL "
*70 E$="WORD"
*80 SCRATCH:1
*90 WRITE:1,A$,B$,C$,D$,E$
*100 RESTORE:1
*110 IF END:1 THEN 150
*120 READ:1,V$
*130 PRINT V$;
*140 GOTO 110
*150 END
*RUN
```

The executed program produces the printout

```
EACH STRING HAS A LEADING AND TRAILING CONTROL WORD
```

Rules: A comma or a colon can be used in an IF END:---THEN statement to separate the file designator from the THEN portion of the statement.

Remarks: The IF END:---THEN statement directs the system to go to a designated out-of-sequence statement when no more data remains on the file.

IF MORE:----THEN
or
IF MORE:----GOTO

Purpose: To provide for a means of testing to determine whether at least one valid data element remains on a binary file when reading the file.

Format: IF MORE: <file designator> {THEN | GOTO} <statement number>

Example:

```
*10 FILES ZORRO
*20 K1=1
*30 A$="EACH STRING "
*40 B$="HAS A "
*50 C$="LEADING AND TRAILING "
*60 D$="CONTROL "
*70 E$="WORD"
*80 SCRATCH:1
*90 WRITE:1,A$,B$,C$,D$,E$
*100 RESTORE:1
*110 READ:1,V$
*120 PRINT V$;
*130 IF MORE:1 THEN 110
*140 END
*RUN
```

The executed program produces the printout

```
EACH STRING HAS A LEADING AND TRAILING CONTROL WORD
```

Rules: A comma or a colon can be used in an IF MORE:---THEN statement to separate the file designator from the THEN portion of the statement.

Remarks: If data elements (or string data) of a binary file are exhausted before input list of a READ: or MAT READ: statement is satisfied, the list is filled out by zeros upon program execution.

The IF MORE:---THEN statement directs the system to go to a designated out-of-sequence statement when more data remains to be read on the file.

Purpose: To permit the word pointer for a random binary file to be positioned so that data can be read or written at any point on the file.

Format: SET: <file designator> TO <expression>

Example: Assume random binary file ORKIN is created via the ACCESS system and its size is three blocks (3 x 320 = 960 words).

```
*10 FILES ORKIN
*20 SET:1 TO 620
*30 FOR P=1 TO 36
*40 WRITE: 1,P
*50 NEXT P
*70 FOR K=655 TO 620 STEP -1
*80 SET:1 TO K
*90 READ:1,N
*100 PRINT N;
*120 NEXT K
*130 END
*RUN
```

Upon execution, the program writes the integers 1 through 36 on file ORKIN, beginning at word 620 and ending at word 655. In addition, the contents of words 620 through 655 are verified and the integers (in reverse order) are printed out as follows:

```
36  35  34  33  32  31  30  29  28  27  26  25  24  23  22

21  20  19  18  17  16  15  14  13  12  11  10   9   8   7

 6   5   4   3   2   1
```

Rules: The SET: statement applies to random binary files only.

Remarks: The expression in the SET: statement is evaluated and its integer portion, if greater than or equal to zero, stored in the word pointer of the designated file. If the integer portion is negative, an explanatory error message and program termination result.

MULTIPLE STATEMENTS WITHIN ONE LINE

While each statement of a program must be confined to a single line, the user can make multiple statements within a single line, utilizing one line number. Statements within a line are separated by means of a reverse slant (\). For example, the line

```
*10 A=12\B=37\C=SQR(A+B)\PRINT A,B,C
```

is equivalent to four statements and is identified by line number 10.

If a multiple-statement line is used in a program employing loops or transfers, a transfer can only be made to the first of the multiple statements. For example,

```
*  10 LET N=0

*  20 READ X,Y,Z PRINT X,Y,Z N=N+1 RESTORE

*  30 IF N  5 THEN 20 DATA 1,2,3

*  40 END
```

SAVING TEMPORARY FILES

When the user terminates the session at the terminal with a logoff sequence, the system is scanned for the user's temporary files. The message

```
n TEMPORARY FILES CREATED
```

is issued, n being the number of files. Each temporary file name is listed, followed by a question mark. The user can respond as follows:

1. carriage return -- implies that this file is to be released; pass to the next file if more temporary files exist.

2. NONE -- implies this and all succeeding files are to be released.

3. SAVE filename -- specifies that this file is to be saved as one of the user's permanent files; pass to the next file if more temporary files exist.

SAVING AND EXECUTING OBJECT FILES

The BRN command can be used to save a file in its object (binary) code form and/or execute a program with such a file. Basic forms of the BRN command to achieve these purposes are as follows:

1. BRN = objfile

   The user's current file is compiled and saved as an object file on random file objfile.

2. BRN = catalog/objfile

   Same as item 1 except that catalog/filename structure is used.

3. BRN objfile

   The contents of random file objfile are loaded into memory and executed. Compilation has already been performed.

4. BRN catalog/objfile

   Same as item 3 except that catalog/filename structure is used.

5. BRN filename = objfile

   The file filename is compiled, saved as an object file on random file objfile, and executed.

6. BRN filename = objfile (NO GO)

   The file filename is compiled and saved as an object file on random file objfile. No execution takes place if (NO GO) option is utilized.

For example,

```
BRN JDOE/RACE,R = MYFILE
```

compiles file RACE; RACE is then saved as object file MYFILE and MYFILE is executed.

```
BRN MYFILE
```

executes object file MYFILE.

If a catalog/filename structure is used, a maximum of three levels is permitted. Legal permission combinations for the catalog/filename structure are:

```
READ
WRITE
APPEND
EXECUTE
READ,WRITE
READ, APPEND
```

The user should note that, as a general rule, object programs are not transferable from software release to software release; in which case, the user should recompile before attempting to run a saved objfile.

FILE ACCESS

For the normal time sharing user, all files (programs) are defined by user identification and a unique file name for each set of files. Since the user identification given to the time sharing system on the logon procedure, and the file name (OLD program name), completely define the file for a normal situation, the time sharing system automatically gives the user access to his own files stored by use of the SAVE control command. However, if the user wishes to make use other files (for instance, those saved by another user), it is necessary, to previously have accessed these files. One method of accessing other users files is by a time sharing subsystem called ACCESS. This subsystem allows the time sharing user to access files that have been saved by others, or that have been stored in the file system by means other than the control command SAVE (e.g., batch-world files), and to place these files at the user's disposal for a session at the terminal. If this feature is required, the user must select ACCESS before he goes to the BASIC system. The ACCESS subsystem is described in Section IV.

SECTION X

EDITOR AND RUNOFF

EDITOR SUBSYSTEM FUNCTIONS

The EDITOR subsystem consists of functions that permit the user to perform the following:

1. Build a text file.
2. Append to an existing text file.
3. Edit a text file by additions, deletions, or corrections.

Under the first condition, where no text exists, the EDITOR subsystem transmits the editing response "ENTER" as a result of the NEW command, to the terminal and calls the TSS data collector to issue an asterisk. The asterisk indicates that the subsystem is in the build mode, and only the system commands #AUTO, #TAPE, #LUCID, #RECOVER, and #ROLLBACK are acceptable.

ENTRY TO EDITOR SUBSYSTEM

Following the logon procedure, the user responds to the initial asterisk with EDITOR. A hyphen (-) then appears to indicate the availability of the EDITOR subsystem.

The action the EDITOR subsystem takes upon being called depends on the file accessed. The file accessed can be in one of two possible conditions:

1. The file contains no text, as in the case of a new file to be built, or possibly no text exists in the file accessed by an OLD _file name_ response.
2. Text exists in the file accessed by an OLD _file name_ response.

Under the first condition, wherein no text exists, the EDITOR subsystem transmits the editing response, ENTER, to the terminal and calls upon the Time Sharing System data collector to issue an asterisk. The asterisk indicates that the system is in build mode and system commands or subsystem names are acceptable as text input only. The entry sequence is as follows:

```
*EDITOR NEW
ENTER
*
```

Under condition two, wherein text does exist, the EDITOR subsystem accepts any editing command following the hyphen response. The entry sequence is as follows:

```
*EDITOR OLD filename
-(any editing command)
```

If the user desires to append data to the file filename, the editing command BUILD is entered and an ENTER and asterisk are transmitted to the terminal as in the first condition. The entry sequence is as follows:

```
*EDITOR OLD filename
-BUILD
ENTER
*
```

NOTE: In the first few examples shown, user entries are underscored, as a teaching aid. These underscores are not part of the file and do not appear with entries made at the terminal.

BUILDING OR ADDING TO A FILE

After the message, ENTER, and the initial data collector asterisk, two methods can be used to build (create) or add to a file. Text can be entered from the terminal via the keyboard, or from paper tape if the terminal is equipped with a tape reader.

Entering Text From Terminal Keyboard

At the keyboard, typing of the desired text can begin. After each carriage return, the system types out an asterisk at the beginning of each new line. This asterisk does not appear in the line of text in any printout of the file.

The following rules apply when entering text:

1. Text can be typed using both uppercase and lowercase letters if both are available on the terminal.

2. The desired text is typed immediately following the asterisk. All characters, including spaces, typed after the asterisk appear in the printout of the file.

```
*EDITOR NEW
ENTER
*THIS LINE IS TYPED WITHOUT LEADING SPACES.
*     THIS LINE CONTAINS 5 LEADING SPACES.
*(carriage return)
-PRINT;*
THIS LINE IS TYPED WITHOUT LEADING SPACES.
     THIS LINE CONTAINS 5 LEADING SPACES.

END OF FILE
```

3. To insert a blank line, the space bar and then the carriage return are used.

   As shown in the example, a carriage return immediately following the asterisk terminates the text entry and produces the "-" response. At this point, editing or time sharing commands can be issued.

4. A carriage return is required at the end of every line of text entered and upon completion of text entry.

5. On a keyboard/display type terminal, the first character typed in replaces the asterisk. To terminate text entry and use an editing command, two number signs (##) and a blank following the asterisk are typed.

6. Service functions recognizable with text entry are #AUTO, #TAPE, #LUCID, #RECOVER, and #ROLLBACK.

Line Numbered Files

Line numbers are not required by the EDITOR or RUNOFF subsystems, but line numbered files are required by most of the other time sharing subsystems. The user can employ the EDITOR and RUNOFF functions on line numbered files for later use under another subsystem. The user can supply one to eight numeric characters as the first entries for each line, or line numbers can be supplied automatically by the Time Sharing System by the use of the #AUTO command in the "BUILD" mode. #AUTO can be used as follows:

1. #AUTOMATIC

   Causes the automatic creation of line numbers by the system, at the point at which the automatic mode is entered (or reentered), with line numbers initially starting at 010 and incrementing by 10 (or, on reentry, resuming where the previous automatic numbering left off). These line numbers appear in the terminal copy, and are written in the file, just as though the user had typed them.

2. #AUTOMATIC n,m

   Causes the automatic creation of line numbers, as above, but starting with line number _n_ and incrementing by _m_.

3. #AUTOMATIC ,m
#AUTOMATIC n,

Causes automatic creation of line numbers beginning at 10 and incrementing by m, or beginning at n and incrementing by 10 (on reentry, the line numbering resumes where it left off).

Normally the line number is followed by a blank. Any nonblank, nonnumeric character affixed to the end of the command #AUTOMATIC causes the blank to be suppressed. For example: #AUTONB or #AUTOMATICX.

No commands are recognized while in the automatic mode. The automatic mode is cancelled by giving a carriage return immediately following the issuance of an asterisk and line number by the system. Upon leaving "#AUTO", return is to the EDITOR "BUILD" mode. The user may not use character delete (@) or line delete (CTRL X) to delete characters associated with the generated line number or its associated blank.

Resequencing Line-Numbered Files

The RESEQUENCE command can be used to resequence the line numbers of the current file. The RESEQUENCE command must be utilized while in the "edit" mode of the Text Editor.

The description of the RESEQUENCE command is in Section IV and repeated below for easy reference.

1. RESEQUENCE

   The line numbers of the current file are resequenced. The resequencing begins with line number 10 and continues in increments of 10. If BASIC is the selected subsystem, the file is resequenced and statement number references in the program are modified correspondingly (GOTO, GOSUB, IF, ON, Print USING). If FORTRAN or no system was selected, statement number references are not affected.

2. RESEQUENCE n,m,x-y

   The line numbers of the current file are resequenced and modifications made according to the subsystem selection. The resequencing begins with line number n and continues in increments of m.

   x and y are specified only if partial resequencing is desired. x gives the starting point and y the ending point of resequencing, inclusive. A null x field (i.e., -y) specifies from beginning of file to line y, and a null y field (i.e., x-) specifies from line x to the end-of-file.

   In general, any blanks preceding a line number are stripped off. Unnumbered lines are accepted, except under the BASIC subsystem, and will have line numbers added, as implied or specified in the command. Care should be taken in resequencing concatenated BASIC files as line numbers are also statement numbers, and statement references, after resequencing, may become invalid.

3. RESEX n,m

Line numbers are inserted at the beginning of each line in the current file, regardless of whether or not line numbers already exist. The numbering begins with n and increments by m, or optionally, begins with 10 and increments by 10, if _n,m_ are not specified. If the first character of the existing line is a numeric, a blank is inserted following the generated line number. If the first character of the existing line is not numeric, no blank is inserted.

4. RESE# n,m

Line numbers are inserted at the beginning of each line in the current file, even if line numbers already exist. This numbering begins with n and increments by m, or optionally begins with 10 and increments by 10 if _n, m_ are not specified. If the first character of the existing line is a numeric, a number sign (#) is inserted following the generated line number. If the first character of the existing line is not numeric, the pound sign is not inserted.

CAUTION: When resequencing, or performing a partial resequence, it is possible to produce files with line numbers out of order. This may be caused by incorrect parameters on partial resequence or when new line numbers exceed eight digits (in non-BASIC files). When line numbers are too large, a warning is given. In either case, recovery may be made by resequencing the total file using a smaller beginning line number or a smaller increment.

Entering Text From Paper Tape

A text file can be created on paper tape to be entered into the computer at a later time. To do this, put the terminal in LOCAL, feed enough tape through the tape drive to ensure that there are no unwanted characters, and type the text. A carriage return, line feed, and two rubouts must follow every line of text. An X-OFF (or DC3) must indicate completion of the text, followed by two rubout characters which provide a timing factor.

To use a prepared tape, enter the EDITOR subsystem, and type #TAPE following the initial asterisk. When the READY response appears, put the prepared tape in the terminal's tape reader and turn on the tape drive. The terminal must be in the online mode.

Input from the tape is accepted until the terminal operator stops the reader, the tape runs out or jams, or an X-OFF (or DC3) character is read from the tape. As the tape is being read, a copy of its contents is printed out on the terminal. When tape input is complete, the system looks for an X-OFF prior to transmitting a carriage return and printing an asterisk. At this time, additional text may be entered at the keyboard or a carriage return can be given to obtain the "-" response and allow editing or printing.

```
*EDITOR NEW
ENTER
*#TAPE
A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK. EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
AS DIRECTED BY THE PROGRAM.

A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.
(X-OFF or DC3)
*(carriage return)
-
```

The #TAPE command can also be used to add text from paper tape to a text file that has been built in a current session at the terminal or has been previously saved (refer to the SAVE and OLD command descriptions in Section IV).

```
-BUILD
ENTER
*#TAPE
(Read from paper tape as described above.)
(X-OFF or DC3)
*(carriage return)
-
```

The #LUCID request is substituted for #TAPE for non-ASCII paper tape input.

A printout of the file shows text from paper tape appended to the original text.

Text from paper tape can be inserted into a file at any point in the file. Refer to the description of ENTER under "Responses From EDITOR" in this section.

PROTECTING FILES

An automatic terminal disconnect, a computer or communication lines malfunction, or a user simply forgetting to save a file before shutting down can cause the loss of input if the user is building or adding to a file. A large file requiring many hours or even days of typing input may be lost. The following paragraphs describe methods of preventing such losses.

The simplest way to ensure against loss from any condition except computer system malfunction is to save portions of the file at intervals while building. In this way, only the last unsaved portion of the file would be subject to loss. (See the following example.)

```
*EDITOR NEW
ENTER
*text ---------
*text ---------
*
*
*
*
*text ---------
*(carriage return)
-SAVE EXAM.1
DATA SAVED--EXAM.1
-
```

(At this point, the editing commands can be used to print or change the file. For each succeeding save, use the RESAVE function and specify the original name. If you wish to continue building this file, use the BUILD command.)

```
-BUILD
ENTER
*

Request for editing function
Request for SAVE
Verification of SAVE
```

NOTE: The use of commands #RECOVER, #ROLLBACK, OLDP, and OLDP# can provide the user with additional means of file protection. Refer to the Section IV for details of use of these commands.

A paper tape of the file contents also provides a hard copy backup in case a file must be rebuilt. This tape can be punched as you build the file from the keyboard. The tape will contain the asterisk printed by the system at the beginning of each line and any lines which were deleted or corrected while building. (See the following example.) If it is necessary to rebuild this file via tape, the rebuilt file must be edited.

```
*EDITOR NEW
ENTER
*#TAPE
READY
*HUMAN LANGUAGES ARE IMPRACTICAL FOR PREPARING
*COMPUTER PROGRAMS BECAUSE THESE LANGUAGES
*CONTAIN MANY AMBIGUITIES AND REDUNDANCIES;
*THE COMPUTER INTERPRETS LANGUAGE ABSOLUTELY
*LITERALLY. BY THE SAME TOKEN, MACHINE
*LANGUAGES ARE ALSO IMPRACTICAL BECAUSE THEY ARE
*DIFFICULT FOR PEOPLE TO USE. MOST PROGRAMMING
*LANGUAGES ARE COMPROMISES BETWEEN HUMAN AND
*MACHINE LANGUAGES.
(X-OFF or DC3)
*(carriage return)
-
```

To create a tape that does not require extensive editing, build a portion of the file, enter the editing function, give a PRINT command, and punch the tape as the file is being printed out. The following example illustrates this method.

```
*text
*text
*(carriage return)
-PRINT;*    (Do not type a carriage return until the tape drive has
            been turned on and the following done:
            (1)  To ensure a clean tape, repeat the rubout character
            until you have a tape leader long enough to be placed in
            the tape drive.
            (2)  Backspace the tape once so that the carriage return
            is wiped out by the last rubout character.
Type a carriage return.
```

The contents of the file is typed out while the tape is being punched. The message END OF FILE is punched into the tape. If the file must be rebuilt via this tape, this message must be deleted.

SEARCH POINTER CONVENTIONS

Each file upon entering EDITOR has a search pointer associated with it. This pointer is located at the beginning of the file until the first editing command is given and is backed up a specified number of lines or returned to the beginning of the file by the BACKUP command. The pointer always points to the beginning of a line, never to a point within the line. This allows several edit operations to be performed on the same line, as long as the operation does not move the search pointer.

The rules governing the movement of the search pointer are as follows:

1. The PRINT, INSERT, REPLACE, DELETE, FIND, CUT, COPY, and PASTE commands cause the search pointer to move forward toward the end of the file, unless the command affects only the line at which the pointer is already located (usually a command with no operand field).

2. Following the execution of any of the commands listed in rule 1., the pointer is located at the last line affected by the command.

3. The BUILD command positions the search pointer to the end of the file. Exiting from the BUILD repositions the search pointer to the beginning of the file.

4. The BACKUP command moves the search pointer backward to the beginning of the file or a specified number of lines from wherever it is located.

5. For commands involving a search operation--a string field is specified--the file is always searched starting at the current location of the search pointer; the search is terminated either by a successful comparison with the specified string field or by encountering the end of the file. In the latter case, the pointer must be backed up before any further editing operations may be performed.

   NOTE: In the line mode, the search pointer can be moved forward or backward by the use of +n or -n with a search verb. "n" is the number lines to move forward (+) or backward (-).

If a given line has already been passed by the search pointer, the BACKUP command or a command with a -n mode must be used to backup the pointer to the line to be operated on.

The current position of the search pointer can always be determined by using a PRINT command with no operand field.

The position of the search pointer is also affected by use of the terminal "break" key to halt the file printout process. The position of the search pointer at the time the break key is pressed is dependent upon the system interfaces. If internal procedures have not been completed when the EDITOR subsystem is notified, the search pointer is positioned back at the last "-" response. If the internal procedures have already been completed prior to transmitting the - response to the terminal, then probably the search pointer position and command execution is as if the break has not occurred.

The following symbols are used in some examples illustrating the location of the search pointer:

<>Location of search pointer at the start of command execution.

>Location of search pointer at the finish of command execution.

<|>Location of search pointer at both start and finish of command execution.

EDITOR LANGUAGE

The EDITOR language is composed of editing commands given by the user while working with a file and responses from the EDITOR subsystem to the user.

Command Format

An EDITOR command may be a single verb only or a verb plus modifier. The modifiers specify variations from the standard operation of the verb and make up the "operand field" of the command. In the examples of command format below, everything following the verb is part of the operand field and, therefore, optional. When the operand field is used, the punctuation shown is required. No intervening blanks are permitted in the command format. (Capitalization of the verb is not required; it is done here to illustrate format.)

```
VERB
VERBm;r
VERBm:st
VERBm:st+st
VERBm:st-st
VERBm:st;r
VERBm:st,st
VERBm:st:st
VERBm:st,st;r
VERBm:st;r:st
```

Where:

m is the mode indicator or +/-
r is the repeat field
st is the string field

An abbreviated form of some verbs is allowed; the abbreviation is the initial letter of the verb.

BACKUP or B
COPY
CUT
DELETE or D
FIND or F
INSERT or I
PASTE
PRINT or P
REPLACE or R
CASE

The following verbs cannot have an operand field:

LINE or L	RUNOFF	STANDARD
STRING or S	VERIFY	
BUILD	NOVERIFY	

The use of the verbs and operands are fully explained and illustrated later in this section. The restrictions and usage rules that apply to the operand field are explained in "Operand Field of the Command" below.

The EDITOR responds to the commands with messages that inform the user when a command has been executed, a mistake in command format has been made, or the end of the file has been reached. These messages are described in "Responses from EDITOR," in this section.

Operand Field Of The Command

The operand field of the command can contain one or more of the following:

- o Mode indicator (used only when a string field is used)
- o Plus (+)n or minus (-)n to move line pointer forward or backward (line mode only). Not applicable to BACKUP(B).
- o String field, preceded by a colon
- o Repeat field, preceded by a semicolon

If more than one of these items is used in a single command, the order must be as shown above.

Insertion or replacement text can also be a part of the operand. Refer to INSERT and REPLACE commands in this section.

The letter V appended to a command results in command verification. Refer to VERIFY and NOVERIFY commands in this section.

The letter B appended to the INSERT command permits insertions immediately before instead of after a specified line or string. Refer to the INSERT command in this section.

Two special forms of the operand are used with the FIND, PRINT, CUT, COPY, PASTE, DELETE, INSERT, and REPLACE commands to obtain AND and OR functions; refer to these command descriptions in this section for details.

MODE INDICATOR

The mode indicators used with the EDITOR verbs are "S" for string mode, #NO and IGNORE (for line numbered files), and "L" for line mode. The mode determines the type of operation to be performed and the interpretation of the string field.

In the line mode, a command acts on one or more complete lines of the file.

```
-PRINT
AA COMPUTER PROGRAM IS A SET OF INSTRUCTIONS
-REPLACE:/AA/
ENTER
*A
*(carriage return)
-PRINT
A
-
```

The entire line was replaced by the single character entered. To correctly perform this function in line mode, the entire line should be retyped as follows:

```
-REPLACE:/AA/
ENTER
*A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS
*(carriage return)
-
```

In string mode, a command acts on a specified string or strings of characters.

```
-PRINT
A COMPUTER PROGRAM IS AA SET OF INSTRUCTIONS
-REPLACES:/AA/
ENTER
*A
*(carriage return)
-PRINT
A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS
-
```

Line mode is the normal mode of operation for EDITOR unless the string mode is specified by the user. When the S mode indicator is added to the verb, only that command is affected. All subsequent commands not having a mode indicator are in line mode.

It is unnecessary to use the L mode indicator except when the command, STRING, has been used. (See "EDITOR Commands" below.) The STRING command causes all commands following to operate in the string mode until the user reverts to the line mode by using a LINE command. If a line mode operation is desired following STRING, the L mode indicator can be added to the verb or, if no string field is given, the command operates in line mode.

In line mode, the search for a successful comparison to the string field (if used) is limited to the initial characters of each line. The characters specified in the string field must correspond to those at the beginning of a line, always starting with the first printing position. In the character-by-character comparison process involved in searching, the first character of the string field is compared with the first character of a line. If these initial characters are not the same, the search proceeds directly to the first character of the next line, rather than to succeeding characters in the same line.

In string mode, a command can act upon any string of characters in the file, regardless of line breaks (carriage returns). The character string acted upon can range from one or more characters within a line, to any number of consecutive characters extending through several lines. For example, a complete sentence that begins in the middle of one line and ends within another line could be deleted without disturbing the rest of the first and last lines. The character string specified in the string field need not be unique to the beginning of a line, of course, but the whole line is not implied, as in line mode -- only the string actually specified.

STRING FIELD

The string field is always preceded by a colon. It consists of a string of characters bounded on both ends by a delimiter. The delimiter can be any character except a blank, control character, or a character contained within a string, and is not entered in the file.

The general form of the string field is:

:dxxxxxxxd

where d is the delimiter and x is any character other than d.

Examples of simple string fields are:

(1) :/THE COMMITTEE/
(2) :.WHEN THE VALUE OF (F/X)*N IS 0,.
(3) :XMR. PERRY, MOR. SMITH, AND MISS JONESX

The delimiter in example (1) is the slant, in example (2) the period, and in example (3) the character X. In all succeeding examples, the slant is used as the delimiter unless it conflicts with a slant in the string, as in example (2). The string to be acted on must be contained within one line.

When the string to be acted on is very long (it can include several lines), two string fields, separated by a comma, can be used. This is called a point-to-point string field. The first string field must uniquely identify the starting point and the second string field must identify the stopping point. Each string field must be contained within a line.

The above examples are shown below in the point-to-point form:

(1) :/TH/,/TEE/
(2) :.WH.,.0,.
(3) :XMRX,XESX

The following conventions concerning the effect of blanks and carriage returns are used by EDITOR in searching:

- o Carriage returns can not be used for comparison purposes.
- o Consecutive blanks in the file must be matched exactly by the blanks in the operand string field: n consecutive blanks in the string field means n consecutive blanks in the corresponding position of the character string in the file.

Certain commands permit two special forms of the string field to be used. String identifiers can be combined by the use of the Boolean AND or OR functions.

The searching conventions must be remembered when specifying string fields for searching to successfully locate the desired portion of the file.

REPEAT FIELD

The repeat field specifies the number of times an operation is to be repeated. The field is always preceded by a semicolon and can contain a number which must be less than 262144 or an asterisk. The number indicates the number of repetitions wanted; the asterisk (*) will result in the operation being repeated throughout the entire file, unless the operation exceeds 262144 occurrences. If the operations exceeds 262144 occurrences, the operation will have to be repeated.

When the repeat field is used without a string field, the operation is always performed in the line mode.

The effect of the repeat field is explained in the detailed descriptions of each command (see "EDITOR Commands" in this section). However, a few brief examples are given below.

PRINT;5 (Prints five lines, beginning at the location of the search pointer.)

PRINT:/YOU/;3 (Prints the first three lines encountered beginning with the characters YOU. This would include YOUR, YOURS, YOU'RE, etc.)

PRINTS:/YOU/;3 (Prints the lines containing the first three occurrences of the characters YOU. This results in three lines of print, possibly the same line repeated if all three occurrences are in the same line.)

PRINT;* (Prints the complete file from the location of the search pointer.)

LINE LENGTH

When creating or verifying lines, the text EDITOR has a buffer restriction. Total line length cannot exceed 1268 characters. When verifying, line length cannot exceed 596 characters. Attempting to exceed these limits may give unpredictable results.

The difference between string and line mode requirements is as follows:

LINE	STRING
The line field must contain the initial characters of the line as it appears in the file.	The string field can contain any characters in the line. (Caution: If the string to be operated upon appears twice in the same line and the second occurrence is where the change is to be made, be sure to include enough characters from the preceding or following word to make the string unique.)

#NO MODE

The #NO mode allows a user to print a line-numbered file without printing the line numbers. This mode is reset by typing #YES.

IGNORE MODE

The IGNORE mode allows the user to disregard line numbers while making modifications to a file using string functions. During execution, each line is scanned commencing with the first character of the line. The first nonnumeric character encountered is established as the first character of the line. To reset the mode to normal, the user types NOIG.

CAUTION: When the first character of the line is a numeric, some nonnumeric character should be inserted following the line number.

Responses From EDITOR

- (hyphen)

The last command has been executed and EDITOR is ready to accept either another EDITOR command or a Time Sharing System command.

ENTER

This response to a REPLACE, INSERT, or BUILD command informs the user that the replacement, insertion, or additional text can now be entered.

An asterisk appears after the ENTER response, indicating that the time sharing data collector now accepts text entry.

LIMIT REACHED

This message occurs only when a repeat field is used with an INSERT or REPLACE command and the text being entered exceeds the buffer capacity. All text input before the LIMIT REACHED message is entered into the file as many times as specified by the last repeat field. The search pointer will be at the last location altered.

To continue inserting or replacing text, back up and find the starting point, repeat the REPLACE or INSERT command and continue entering the rest of the text.

END OF FILE

This message occurs when the search pointer has reached the end of the file. This is the normal response to a command with an * in the repeat field. It also occurs when the specified string field does not appear in the file.

Following this message, a BACKUP command should be given if more work is to be done on the file.

COMMAND UNKNOWN

EDITOR does not recognize the command, either because it is illegal or because it is misspelled. This response may cause the EDITOR search pointer to be repositioned to the beginning of the current file. To return to the place in the file where the faulty command was given, the user can make use of the FIND command.

STRING ERROR

The command contains one or more of the following errors:

1. The string mode has been specified but no string field has been entered.

2. The ending delimiter is missing.

3. One or more characters have been typed on the same line following the final delimiter.

REPEAT ERROR

The repeat field contains a character other than a number or *. Retype the command correctly.

END OF FILE - REQUEST EXECUTED XX TIMES

The above message occurs when a repeat field is used and the repeat field specified is greater than the number of occurrences in the file or the repeat field is an asterisk. XX represents the number of times the specified function was executed.

PASTE NOT EXECUTED, NO DATA

The above message occurs as a result of one of two reasons:

1. The user failed to cut or copy data prior to issuing a PASTE command

2. A system malfunction occurred preventing the data specified from being cut or copied.

TEXT INSERTION ERROR

This message occurs as a result of a missing delimiter or text following the terminating delimiter.

OPERAND ERROR

This message occurs as a result of an operand error. Either an inappropriate operand was used, an operand was utilized where operands are not permitted, or an operand was expected and not found.

UNABLE TO CUT/COPY, NO FILE SPACE

The message indicates an inability to cut or copy due to a lack of temporary file space to store the cut or copied data.

CUT/COPY TRUNCATED, PERFORM PASTE TO CONTINUE

This message occurs as a result of an extensive amount of text being cut or copied, causing the cut/copy file to overflow. Performing a paste function following this message allows continued use of cut/copy file.

<50> WORK FILE, TABLE FULL, STATUS 36
<50> WORK FILE, SYSTEM LOADED, STATUS 40
<50> WORK FILE, STATUS 10
<52> CURRENT FILE NOT DEFINED
<50> NO FILE SPACE, STATUS 13

RUNOFF Format Control Words

RUNOFF format control words can be entered in the text file during the building or editing phase of the EDITOR subsystem to achieve such text format as spacing, indentation, and page numbering. These format control words remain embedded within the text file but are removed in a printout of the file by way of the RUNOFF subsystem command REFORM. Refer to Section IV for descriptions of RUNOFF subsystem commands and format control words.

TIME SHARING SYSTEM CONTROL COMMANDS

Time Sharing System control commands perform nonediting functions (e.g., saving or purging files, calling in other subsystems) for EDITOR. These control commands can be entered immediately after the appearance of the - response, the "-" indicating system readiness to accept a command. Time Sharing System control commands and their application to the EDITOR subsystem are described in the Section IV.

EDITOR COMMANDS

The EDITOR commands are described below in the following order (note the permissible abbreviations):

AFTLIN/BEFLIN or A/BEFL
BACKUP or B
BUILD
CASE/STANDARD
COPY
COLUMN or COLS
CUT
DELETE or D
FIND or F
INSERT or I
LIMIT
LINE
MARK
MASK
MODE or M
OCCURRENCE or O
OCTAL or OCTL
PASTE
PRINT or P
REPLACE or R
RUNOFF
STRING
TRANSPARENT or T
VERIFY/NOVERIFY
WHERE or W

EDITOR commands can be employed singly or in multiples, the only restriction being that the one or more commands be contained on a single line. With use of the single command, a "-" response is issued upon command execution and control is returned to the user. With multiple command use, the series of commands are executed before the "-" response is issued and control returned to the user. Commands (and accompanying operands, if any) in a multiple command line must be separated by one or more blanks. For example,

```
-F    P;5    B;5    P;3    D;5
```

An unsuccessful command execution in a multiple command aborts the execution of any remaining commands.

NOTE: The slant is used as a delimiter to illustrate EDITOR commands below. Any keyboard character, except a blank or control character, can be used as a delimiter.

AFTLIN Command And BEFLIN Command

AFTLIN and BEFLIN are acronyms for "AFTer LINe" and "BEFore LINe." They allow the user to append data at the end of the line(s) or at the beginning of the line(s).

AFTLIN (short form A) command allows the user to append data to a line or number of lines specified in the repeat field. Input data can be entered in either of two forms.

1. Input data can follow the repeat factor (;n or ; *) in form ":/input data/", example:

```
-A;1:/abcdef/
```

where the string "abcdef" will be appended to the current line (;1). If the repeat factor (;n) is greater than one, the string "abcdef" will be appended to the current line plus the next n -1 lines.

2. Input data can follow the ENTER message.

```
-A;1
ENTER
*abcdef
*CR(carriage return)
```

NOTE: In the above form, the numeric "1" specifies the current line. If more than the specified "n" lines of input is entered, the excess is ignored.

If ";*" appears in the repeat field, the input data is appended to all of the remaining lines. For example:

```
-A;*
ENTER
*001abc
*002def
*003ghi
*CR(carriage return)

END OF FILE - REQUEST EXECUTED nnn TIMES
```

In the above example, input line "001abc" is appended to the current line, input "002def" is appended to the current line plus one, and "003ghi" is appended to the current line plus two. In this form the repeat field is ignored.

BEFLIN, is not permitted a short form since "B" would conflict with the verb BACKUP; the abbreviation BEFL is permitted. The same rules apply to BEFLIN as to the AFTLIN, except the input data is prefixed to the beginning of the line.

BACKUP Command

The BACKUP command moves the search pointer backward the number of lines specified in the operand field. If the operand field is blank, the pointer backs up to the beginning of the file. The use of BACKUP is illustrated in the examples given for other EDITOR commands.

The formats and execution are as follows:

Command	Execution
BACKUP	Backup search pointer to beginning of file.
BACKUP;n	Backup search pointer n consecutive lines.

BUILD Command

The BUILD command enables the user to append additional text to his text file. When the user gives the BUILD command, the EDITOR subsystem positions the search pointer to the end of the file and responds with ENTER.

The text to be entered is typed on lines following the ENTER response. When text entry is comple e, a carriage return only in response to the asterisk generates the - response.

The text entered following the ENTER response is appended to the file. When the carriage return is given to signal the end of text entry, the - response is given and the search pointer is returned to the beginning of the file.

```
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER, PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE SAME TERMINAL.

END OF FILE
BUILD
ENTER
*THE PROGRAM CAN BE CORRECTED OR CHANGED BY
*THE USER AS IF HE WERE CONVERSING BY PHONE,
*EXCEPT IN THIS CASE, THE CONVERSATION IS
*TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
*OF TERMINAL IN USE.
*(blank,carriage return)
*IF THE PROGRAM CONTAINS A MISTAKE, THE
*COMPUTER INFORMS THE USER.
*(carriage return)
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER, PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE SAME TERMINAL.
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.

IF THE PROGRAM CONTAINS A MISTAKE, THE
COMPUTER INFORMS THE USER.

END OF FILE
```

CASE Command And STANDARD Command

The CASE command enables the user to set the mode of the EDITOR subsystem so as to permit it to search a dual-case (upper and lower) text file from a terminal with a single-case keyboard.

The STANDARD command removes the EDITOR subsystem from the CASE mode.

The formats and execution of the CASE command are as follows, d representing a delimiter other than that used for delimiting string fields. The delimiter must be a character not contained within the file.

Command	Execution
CASE	String fields of commands cause search and location by text, ignoring case.
CASE UPPER d	String fields of commands cause search and location of uppercase text. The specified delimiter denotes uppercase text for display or insertion.
CASE LOWER d	String fields of commands cause search and location of lowercase text. The specified delimiter denotes lowercase text for display or insertion.

For example, a line of text in a file consists of the following:

EDITOR and RUNOFF are subsystems of TEXT EDITOR.

The user is restricted to an uppercase terminal.

```
-CASE
-PRINT:/EDITOR/
EDITOR AND RUNOFF ARE SUBSYSTEMS OF TEXT EDITOR

-CASE UPPER $
-PRINT
$EDITOR$ AND $RUNOFF$ ARE SUBSYSTEMS OF $TEXT$ $EDITOR$

-REPLACES:/SUBSYSTEMS/:/$S$UBSYSTEMS/
```

The line of text in the file now consists of the following:

EDITOR and RUNOFF are Subsystems of TEXT EDITOR.

Note that the specified delimiter does not become part of the text.

COPY Command

The COPY command allows the user to copy a specified portion of text and hold it in reserve for a PASTE command. The copied text is not removed from the file but is repeated at the location specified by PASTE. Several sequential COPY commands can be given and the collected text inserted with a single PASTE command. Examples of the use of COPY are included with the PASTE examples.

The format and execution are as follows:

Command	Execution
COPY	Copy the line at which the search pointer is currently located. (A repeat field can be used with this form.)
COPY:/xxx/	Copy line identified by xxx.
COPY:/xxx/;n	Copy the next n lines identified by xxx. (* can be used to copy all such lines.)
COPY:/xxx/,/yyy/	Copy the block of lines starting with the line identified by xxx through the line identified by yyy. (A repeat field can be used to copy n or all such blocks of lines.)
COPYS:/yyy/	Copy the line that contains the specified string.
COPYS:/yyy/;n	Copy n occurrences of the line that contains the specified string. (* can be used to copy all such lines.)
COPYS:/yyy/,/zzz/	Copy text between points yyy and zzz, inclusive. (A repeat field can be used with this form also.)
COPY:/st1/+.../stn/	Copy line containing all of the specified (a maximum of five) strings. (A repeat field can be used to copy n or all such lines.)
COPY:/st1/-.../stn/	Copy line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to copy n or all such lines.)

Two special forms of the operand are permissible with the COPY command to identify lines containing specified strings. These forms of the command are referred to as the Boolean AND and OR functions. The operand can consist of up to five strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the copy. For example, with d representing a delimiter, the format is

COPY:dSTRING1d+....+dSTRING5d

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--any one of the listed strings need be present to achieve the copy. For example, with d representing a delimiter, the format is

```
COPY:dSTRING1d-...-dSTRING5d
```

Note that these two special forms of the operand are equivalent in line or string mode.

COLUMN Function

The function COLS:(xxx-yyy) allows the user to restrict string searches and modifications in a horizontal direction; i.e., to a specific range of character positions within one or all lines (depending on commands used). It is particularly useful if data is in columnar (tabular) format. For example:

```
-COLS:(9-11)   (User restricts the horizontal range to the
               characters located within columns 9 through
               11 (3 characters) inclusive.)

-P;2           (Print two lines)

1234567890123456789012345678900.........
abc def abc def abc def abc..............

-RS:/abc/:/xyz/        (Replace the string abc with xyz)

-P             (Print the current line)

abc def xyz def abc def abc.............
```

NOTE: The string "abc" in columns 1, 2, and 3 was not affected since column function was restricted to columns 9, 10 and 11. A repeat factor is acceptable.

Limitations:

1. Character string must start in the first column specified and terminate in the last column specified.
2. Multiple occurrences of strings within column parameters are not permitted.
3. Multiple column parameters are not permitted.
4. Only numerics are permitted within parentheses; use of characters other than numerics result in error messages.

NOCO nullifies the column function.

CUT Command

The CUT command performs the same functions as COPY, except that the copied text is deleted from its present location. Examples of this are included with the PASTE examples. The formats and execution are as follows.

Command	Execution
CUT	Copy and remove the line at which the search pointer is located. (A repeat field can be used with this form.)
CUT:/xxx/	Copy and remove the line identified by xxx.
CUT:/xxx/;n	Copy and remove the next n lines identified by xxx. (* can be used to copy and remove all such lines.)
CUT:/xxx/,/yyy/	Copy and remove the block of lines starting with the line identified by xxx through the line identified by yyy. (A repeat field can be used to copy and remove n or all such blocks of lines.)
CUTS:/yyy/	Copy and remove the line that contains the specified string.
CUTS:/yyy/;n	Copy and remove n occurrences of the line that contains the specified string. (* can be used to copy and remove all such lines.)
CUTS:/yyy/,/zzz/	Copy and remove text between points yyy and zzz, inclusive. (A repeat field can be used to copy and remove n or all such occurrences of text.)
CUT:/st1/+.../stn/	Copy and remove the line containing all of the specified (a maximum of five) strings. (A repeat field can be used to copy and remove n or all such lines.)
CUT:/st1/-.../stn/	Copy and remove the line or the lines containing any one of the specified (a maximum of five) strings. (A repeat field can be used to copy and remove n or all such lines.)

Two special forms of the operand are permissible with the CUT command to identify lines containing specified strings. These forms of the command are referred to as the Boolean AND and OR functions. The operand can consist of up to five strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the cut. For example, with d representing a delimiter, the format is

CUT:dSTRING1d+....+dSTRING5d

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the cut. For example, with d representing a delimiter, the format is

CUT:dSTRING1d-...-dSTRING5d

Note that these two special forms of the operand are equivalent in line or string mode.

DELETE Command

The DELETE command allows the user to delete any number of characters, words, or lines from his file. The operand field of the command specifies the text to be deleted. If no operand field is given, the line where the search pointer is located is deleted.

The formats and execution are as follows:

Command	Execution
DELETE or DELETE;n	Delete the or lines at which search pointer is currently located.
DELETE:/xxx/	Delete the line identified by xxx.
DELETE:/xxx/;n	Delete the next n lines identified by xxx. (* can be used instead of n to delete all such lines.)
DELETE:/xxx/,/yyy/	Delete the block of lines starting with the line identified by xxx through the line identified by yyy. (A repeat field can be used to delete n or all such lines.)
DELETES:/yyy/	Delete a specified string.
DELETES:/yyy/;n	Delete n occurrences of a specified string. (* can be used instead of n to delete all such occurrences.)
DELETES:/yyy/,/zzz/	Delete text between points yyy and zzz, inclusive. (A repeat field can be used with this form also.)
DELETE:/st1/+.../stn/	Delete the line containing all of the specified (a maximum of five) strings. (A repeat field can be used to delete n or all such lines.)
DELETE:/st1/-.../stn/	Delete the line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to delete n or all such lines.)

To delete a string of characters, use DELETE in the string mode with or without a repeat field.

```
-PRINT
(HAVE ALL INSTRUCTIONS EXECUTED O) ON A COMPUTER.
-DELETES:/ O/
-PRINT
(HAVE ALL INSTRUCTIONS EXECUTED) ON A COMPUTER.
-
```

To delete from point-to-point, use delete in the string mode with two string fields and with or without a repeat field. All data between and including the two points indicated is deleted.

```
-PRINT;4
COMPUTER PROGRAMS BECAUSE THESE LANGUAGES
CONTAIN MANY AMBIGUITIES AND REDUNDANCIES;
THE COMPUTER INTERPRETS LANGUAGE ABSOLUTELY
LITERALLY.  BY THE SAME TOKEN, MACHINE
-B
-DS:/THE C/,/.  /
-B
-P;4
COMPUTER PROGRAMS BECAUSE THESE LANGUAGES
CONTAIN MANY AMBIGUITIES AND REDUNDANCIES;
BY THE SAME TOKEN, MACHINE
LANGUAGES ARE ALSO IMPRACTICAL BECAUSE THEY ARE
-
```

To delete one or more lines, use DELETE in line mode, with or without a string field and/or repeat field. If both a string field and repeat field are used, the indicated number of lines beginning with the specified string are deleted. If no string field is used with the repeat field, the indicated number of lines is deleted, beginning at the location of the search pointer.

```
-PRINT;4
HUMAN LANGUAGES ARE IMPRACTICAL FOR PREPARING
COMPUTER PROGRAMS BECAUSE THESE LANGUAGES
CONTAIN MANY AMBIGUITIES AND REDUNDANCIES;
THE COMPUTER INTERPRETS LANGUAGE ABSOLUTELY
-B;3
-D;3
-PRINT;2
THE COMPUTER INTERPRETS LANGUAGE ABSOLUTELY
LITERALLY.  BY THE SAME TOKEN, MACHINE
-
```

To delete all lines having a common beginning, use DELETE in line mode with a repeat field. Note that in the following example the sentence "ALL LANGUAGE INSTRUCTION MUST BE" is not deleted because the letter A is preceded by blanks.

```
-PRINT;*

COMPUTER PROGRAMS

A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK.  EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM.  IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
AS DIRECTED BY THE PROGRAM.

A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.

     THE PROGRAM MUST BE SUBMITTED TO THE
     COMPUTER IN A LANGUAGE THAT THE
     COMPUTER RECOGNIZES.

     ALL LANGUAGE INSTRUCTION MUST BE
     COMPLETE AND BE PRECISELY STATED.

END OF FILE
B
-DELETE:/A/;*

END OF FILE - REQUEST EXECUTED    3 TIMES
B
-PRINT;*

COMPUTER PROGRAMS

TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK.  EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM.  IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION

BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.

     THE PROGRAM MUST BE SUBMITTED TO THE
     COMPUTER IN A LANGUAGE THAT THE
     COMPUTER RECOGNIZES.

     ALL LANGUAGE INSTRUCTION MUST BE
     COMPLETE AND BE PRECISELY STATED.

END OF FILE
```

Two special forms of the operand are permissible with the DELETE command to identify lines containing specified strings. These forms of the command are referred to as "Boolean AND and OR functions." The operand can consist of up to five strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the delete. For example, with d representing a delimiter, the format is

DELETE:dSTRING1d+....+dSTRING5d

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the delete. For example, with d representing a delimiter, the format is

DELETE:dSTRING1d-...-dSTRING5d

Note that these two special forms of the operand are equivalent in line or string mode.

FIND Command

The FIND command moves the search pointer through the file. FIND may be used with or without an operand field.

If in doubt as to where the search pointer is located, give the PRINT command with no operand field. The resulting printout is the line pointed to by the search pointer. It is advisable, when editing a file in which the specified string may appear more than once, to print the line before changing the file, in order to ensure that the change is made in the right place.

The repeat field can be used with a string field in the FIND command. The search and comparison continues until the comparison is made as many times as indicated. When execution is completed, the "-" response appears. If the repeat field is used without a string field, the search pointer moves forward n number of lines as indicated by the repeat field.

The formats and execution are as follows:

Command	Execution
FIND	Advance search pointer one line.
FIND;n	Advance search pointer n lines.
FIND:/xxx/	Find the line identified by xxx.
FIND:/xxx/;n	Find nth line identified by xxx.
FINDS:/yyy/	Find the line containing specified string.
FINDS:/yyy/;n	Find the line containing the nth occurrence of the specified string.
FIND:/xxx/+/yyy/+...	Find the line containing all of the specified strings. (A repeat field can be used to find n or all such lines.)
FIND:/xxx/-/yyy/-...	Find the line containing one of the specified strings. (A repeat field can be used to find n or all such lines.)

To find a specified string, not at the beginning of the line, use **FIND** in the string mode.

```
-FINDS:/SUBM/
-PRINT
     THE PROGRAM MUST BE SUBMITTED TO THE
-BACKUP;4
-PRINT;*
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.

     THE PROGRAM MUST BE SUBMITTED TO THE
     COMPUTER IN A LANGUAGE THAT THE
     COMPUTER RECOGNIZES.

END OF FILE
```

To find a string past the point where it next occurs, use **FIND** in the string mode with a repeat field.

```
-PRINT;6
COMPUTER PROGRAMS

A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK. EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY
-B
-FINDS:/IS/;3
-PRINT
TASK. EACH INSTRUCTION IS PERFORMED IN THE
```

To find a specified number of lines, use FIND in line mode with a repeat field. The number in the repeat field includes the line at which the search pointer is located at the beginning of execution (unless FIND is used without a string field, in which case line 1 is the line following).

```
-PRINT;4
THE TIME-SHARING SYSTEM USES A TECHNIQUE BY
WHICH PROGRAMS ARE HANDLED IN PARALLEL.  A
SUPERVISORY PROGRAM ACTS AS A CONTROLLER OF
THESE PROGRAMS, CONTROLLING "STOP" AND "GO"
-FIND;1
-PRINT
SIGNALS TO INPUTS FROM TERMINALS AND
```

Two special forms of the operand are permissible with the **FIND** command to identify lines containing specified strings. These forms of the command are referred to as the Boolean **AND** and **OR** functions. The operand can consist of up to five strings connected by plus signs for the **AND** form and minus signs for the **OR** form. The strings can be in any order; i.e., the fifth string in order of appearance in the line can be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the find. For example, with d representing a delimiter, the format is

```
FIND:dSTRING1d+....+dSTRING5d
```

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the find. For example, with d representing a delimiter, the format is

```
FIND:dSTRING1d-...-dSTRING5d
```

Note that these two special forms of the operand are equivalent in line or string mode.

Examples of the use of these forms of the operand are given with the description of the PRINT command above.

INSERT Command

The INSERT command allows the user to insert any number of characters, words, or lines into his file. The operand field of the INSERT command specifies the point after which the insertion is to be made and can take one of two forms, depending on the length of the text being inserted.

The first list below illustrates the format to be used when the operand field cannot be contained on one line. The system responds to the INSERT command with the word ENTER. The text to be inserted is then typed on lines following ENTER. When text entry is complete, a carriage return following the asterisk generates the "-" response. The second list illustrates the use of INSERT with short strings; the ENTER response is not given in this use of the command.

The formats and execution are as follows:

Command	Execution
INSERT	Insert after the line at which the search pointer is currently located.
INSERT:/xxx/	Insert after the line identified by xxx.
INSERT:/xxx/;n	Insert after each of the next n lines identified by xxx. (* can be used instead of n to insert after all such lines.)
INSERTS:/yyy/	Insert after point yyy.
INSERTS:/yyy/;n	Insert after each of n successive occurrences of point yyy. (* can be used instead of n to insert after all such occurrences.)
INSERT:/stl/+.../stn/	Insert after line containing all of the specified (a maximum of five) strings. (A repeat field can be used to insert after n or all such lines.)

Command	Execution
INSERT:/stl/-.../stn/	Insert after line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to insert after n or all such lines.)

When inserting short strings of text, the following formats can be used.

NOTE: The command and the entire operand field must be on the same line. This format does not accept a carriage return before the final delimiter.

Command	Execution
INSERT:/xxx/:/bbb/	Insert string bbb after the line identified by xxx.
INSERT:/xxx/;n:/bbb/	Insert string bbb after each of the next n lines identified by xxx. (* can be used instead of n to insert after all such lines.)
INSERTS:/yyy/:/bbb/	Insert string bbb after point yyy.
INSERTS:/yyy/;n:/bbb/	Insert string bbb after each of n successive occurrences of point yyy. (* can be used instead of n to insert after all such occurrences.)

To insert one or more lines, use INSERT in the line mode with or without a string field and/or repeat field. If no string field is used, the insertion is made after the line where the search pointer is located. For insertions of more than one line, each new line must be followed by a carriage return to prevent it from running into the next line.

```
-PRINT;6
QUICKLY AT THE SAME TERMINAL.
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.
-B;5
-INSERT
ENTER
*IF THE PROGRAM CONTAINS A MISTAKE, THE
*COMPUTER INFORMS THE USER.
*(carriage return)
-B;3
-PRINT;*
QUICKLY AT THE SAME TERMINAL.
IF THE PROGRAM CONTAINS A MISTAKE, THE
COMPUTER INFORMS THE USER.
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.

END OF FILE
```

To insert a string of characters, use INSERT in the string mode with a string field and with or without a repeat field. The string field must identify the point after which the insertion is to be made.

```
-PRINT;4
THE TIME-SHARING SYSTEM USES A TECHNIQUE BY
WHICH PROGRAMS ARE HANDLED IN PARALLEL.
THUS, TIME-SHARING PERMITS A USER TO WORK
DIRECTLY WITH THE COMPUTER, WHETHER IT IS
-B
-INSERTS:/LEL./
ENTER
* A
*SUPERVISORY PROGRAM ACTS AS A CONTROLLER OF
*THESE PROGRAMS, CONTROLLING "STOP" AND "GO"
*SIGNALS TO INPUTS FROM TERMINALS AND
*PREVENTING DEMANDS OF ONE TERMINAL FROM
*INTERFERING WITH DEMANDS OF OTHER TERMINALS.
*(carriage return)
-B
-PRINT;9
THE TIME-SHARING SYSTEM USES A TECHNIQUE BY
WHICH PROGRAMS ARE HANDLED IN PARALLEL.  A
SUPERVISORY PROGRAM ACTS AS A CONTROLLER OF
THESE PROGRAMS, CONTROLLING "STOP" AND "GO"
SIGNALS TO INPUTS FROM TERMINALS AND
PREVENTING DEMANDS OF ONE TERMINAL FROM
INTERFERING WITH DEMANDS OF OTHER TERMINALS.
THUS TIME-SHARING PERMITS A USER TO WORK
DIRECTLY WITH THE COMPUTER, WHETHER IT IS
-F:/THE PROGRAM/
-PRINT
THE PROGRAM BE CORRECTED OR CHANGED BY
-INSERTS:/RAM /:/CAN /
-P
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
```

To insert at the beginning of the file, use INSERTB in the line mode with no operand field.

```
-PRINT;3
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.
-B
-INSERTB
ENTER
*COMPUTER PROGRAMS
*(blank,carriage return)
*A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
*TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
*TASK. EACH INSTRUCTION IS PERFORMED IN THE
*SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY,
*THE COMPUTER PROCESSES AND PRODUCES INFORMATION
*AS DIRECTED BY THE PROGRAM.
*(carriage return)
-B
```

```
-PRINT;11
COMPUTER PROGRAM
A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELL A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK.  EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
AS DIRECTED BY THE PROGRAM.
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.
```

The INSERT command, in conjunction with the #TAPE command, allows the user to insert text from paper tape in the file at any point in the file. At the selected point (as determined by the operand of the INSERT command), the user activates the paper tape reader to read in the tape after the appearance of the ENTER response. Upon termination of tape read, the user gives a carriage return in response to the asterisk and the - response appears.

```
-INSERT (appropriate operand)
ENTER
*#TAPE
READY

(user activates paper tape reader and
text is read in from tape.)

*(carriage return)
-
```

Text may be alternatively inserted from the keyboard and from paper tape.

```
-INSERT (appropriate operand)
ENTER
*Text entered by user
*more text
*last line of text
*#TAPE
READY
(user activates paper tape reader and text
is read in from tape.)
*Text entered by user
*more text
*last line of text
*(carriage return)
-
```

The INSERT command, as indicated in the descriptions of the command above, provides for insertion of data following the specified line or string. An optional operand, the letter B, can be used with the INSERT command to achieve insertion before the specified line or string.

```
-STRING
-F
-P
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
-INSERTB:/THE/:/THEREFORE, /
-P
THEREFORE, THE PROGRAM  CAN BE CORRECTED OR CHANGED BY
-
```

Two special forms of the operand are permissible with the INSERT command to identify lines containing specified strings. These forms of the operand are referred to as the Boolean AND and OR functions. The operand can consist of up to five separate strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--al of the strings listed must be present to achieve the insert. For example, with d representing a delimiter the format is

INSERT:dSTRING1d+....+dSTRING5d

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the insert. For example, with d representing a delimiter, the format is

INSERT:dSTRING1d-...-dSTRING5d

Note that these two special forms of the operand are equivalent in line or string mode.

LIMIT Function

The LIMIT function allows the user to specify a portion of a line numbered file within which all further verb operations are restricted.

SAMPLE USAGE:

-LIMIT:/203/,/506/ or L:/203/,506/

This mode establishes a subset of a file wherein the line numbered 203 is the first line and line number 506 is the last line. All future function of verbs are executed only within the range specified, i.e., lines which begin with numbers between 203 and 506.

When specifying "LIMIT," if the current line pointer is located outside of the range specified, the pointer will be automatically positioned within the limits range. When returning to the normal "NORM" mode, the line pointer will remain pointing at the last line accessed while in the "LIMIT" mode.

It is possible to insert "BEFORE" or "FOLLOWING" within the specified limited range. If the line numbers of the line(s) inserted are less than or greater than (respectively) the original limit range, the specified limits remain in effect. However, if the line numbers of the line(s) inserted are encompassed within the original limits range, the range is adjusted to include those lines inserted.

NOTE: The LIMIT mode cannot function with Automatic Line Numbering (#AUTO) or RESEQUENCE.

To reset the mode to normal, the user need only type in NORM.

LINE Command

The LINE command counteracts the effect of STRING by placing EDITOR in the line mode, its normal mode of operation. All commands operate in line mode unless the S mode indicator is added to the verb. LINE never has an operand field and is only used to nullify the STRING command.

MARK Command

Whenever a user types MARK, a search of the file is begun for a line commencing with a ".MARK" or ".MARK FILENAME". If a line starting with ".MARK" is located, and a file name is specified, the file is accessed and the data on the specified file will replace the ".MARK" line. If the line does not contain a file name, the user is queried as to the file to be accessed. If a "MARK" line is not found, the user is so informed.

Files accessed utilizing the ".MARK FILENAME" sequence may contain embedded ".MARK" lines. If the "MARK" command (verb) is followed with a repeat of ";*", each time a normal end of file condition is reached following a successful access of a specified "MARK" file, the current file is searched again to ensure that the accessed file did not contain a ".MARK" line.

Limitations:

1. MARK operates in a "NOVERIFY" environment.

2. The "MARK" command cannot be used in conjunction with the "LIMIT" function since "LIMIT" checks to see if the file is line numbered.

3. Catalog/file strings are not permitted, nor are multiple files; e.g., file1;file2,etc.

MASK Function

The MASK function allows the user to manipulate a string without disturbing the surrounding characters. For example:

-MASK # (User sets the "MASK" mode using the number sign as the delimiter)

-P;2 (Print two lines)

```
..........DATANET355..................
.................DATANET305................
```

-B;1 (Back the line pointer up one line)

IVS:/NET###/:2:/?/ (Insert and verify a question mark following the string "NET" followed by any three characters, do it twice)

```
..........DATANET355?...................

.................DATANET305?................
```

Limitation: The mask character is only acceptable in the field containing the string to be worked on. It is not acceptable in the replacement field as a "mask" character.

NOMA nullifies the mask function.

MODE Command

The MODE command allows the terminal operator to determine previously established modes (Verify, String, Line, Case, etc.). The verb "MODE" or short form M can be typed to determine which modes have been set.

OCCURRENCE Function

The use of the "O" operand allows the user to operate on a specific occurrence of a string. The use of the additional repeat field (;n) specifies which occurrence. For example:

Suppose a line contained the following repetitive data:

```
D.....D.....D.....D.....D.....D.....D.....D.....D.....D.....
```

In the above example it would be extremely difficult (if not impossible) to access the sixth occurrence of the string "D" without replacing the entire line.

With the use of the "Occurrence" modifier, replacement of the character would be performed as follows:

```
-RVO:/D/;6:/X/      (Replace and verify the sixth occur-
                    rence of the character "D" with "X")

D.....D.....D.....D.....D.....X.....D.....D.....D.....D.....
```

Suppose the user desired to replace every second occurrence of the character "D" with the character "X" and do it three times. This would be performed as follows:

```
-RVO:/D/;2;3:/X/      (The first repeat field (;2) indi-
                      cates which occurrence, the second
                      repeat field (;3) performs as normal
                      and indicates the number of times)

D.....X.....D.....X.....D.....X.....D.....D.....D.....D.....
```

OCTAL Function

The "OCTL d" function allows the user to designate a unique character (d) to precede an octal number. For example:

```
-OCTL $     (User identifies the dollar sign as the octal
            delimiter to be used)

-P          (Print the current line)

.......on at 9.084 - off at 9.140 on 06/24/75..............

-RS:/at/;2:/$100/     (Replace the string "at" twice with the
                       octal character 100 (@))

-P          (Print the current line)

.......on @ 9.084 - off @ 9.140 on 06/24/75..............
```

The OCTL delimiter is functional within the build mode of the Text Editor providing the mode was set prior to entering BUILD. For example:

```
-OCTL %         (User defines the percent sign as the octal delimiter)

-BUILD          (User enters the BUILD mode)

ENTER           (Text Editor "ENTER" command)

*.......on %100 9.084 - off %100 9.140 on 06/24/75.............
*cr             (User exits the BUILD mode)

-FV;*           (Position to the last line of the file and print)

.......on @ 9.084 - off @ 9.140 on 06/24/75.............

END OF FILE - REQUEST EXECUTED   1 TIMES
```

Caution: No tests are made to determine the validity of the octal character.

NOCT nullifies the octal function.

PASTE Command

The PASTE command inserts the collected CUT or COPY text into the specified location. In order to PASTE the copied text in more than one location, successive PASTE instructions must be used. Once a PASTE command has been executed, the next COPY or CUT command wipes out the previously accumulated COPY or CUT text.

Command	Execution
PASTE	Insert text after the line at which the search pointer is currently located.
PASTE:/xxx/	Insert text after the line identified by xxx.
PASTE:/xxx/;n	Insert text after each of the next n lines identified by xxx. (* can be used to insert after all such lines.)
PASTES:/yyy/	Insert text after point yyy.
PASTES:/yyy/;n	Insert text after each of n successive occurrences of point yyy. (* can be used to insert after all such occurrences.)
PASTE:/st1/+.../stn/	Insert text after all specified (a maximum of five) strings. (A repeat field can be used to insert text after line containing n or all such lines.)
PASTE:/st1/-.../stn/	Insert text after line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to insert text after n or all such lines.)

To cut and paste one or more lines, use CUT in the line mode, with or without a string field and/or repeat field. If both a string field and repeat field are used, the indicated number of lines beginning with the specified string is copied, removed, and then inserted by PASTE. If no string field is used with the repeat field, the indicated number of lines is copied and removed, beginning at the location of the search pointer.

```
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER, PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE SAME TERMINAL.

IF THE PROGRAM CONTAINS A MISTAKE, THE
COMPUTER INFORMS THE USER.
THE PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.

END OF FILE
B
-FIND:/QUICKLY/
-FIND;1
-CUT;3
-PASTE:/OF/
-B
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER, PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE SAME TERMINAL.

THE PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.

IF THE PROGRAM CONTAINS A MISTAKE, THE
COMPUTER INFORMS THE USER.

END OF FILE.
```

To paste the same text in several locations, use CUT or COPY, then successive PASTE commands, one for each insertion needed. The example illustrates a form letter and mailing list contained in the same file. In this case, a continuous PASTE command is used, since each insertion is made following a line beginning with the same word.

```
-PRINT;*

WE TAKE GREAT PLEASURE IN ANNOUNCING
                 .
                 .
                 .
                 .
YOURS VERY TRULY,

COMPANY NAME
ADDRESS
CITY,STATE

MR.  A.  A. ADAMS
ADDRESS
CITY,STATE

DEAR MR. ADAMS:
                 .
                 .
                 .
MR.  X.  Y. ZILCH
ADDRESS
CITY,STATE

DEAR MR. ZILCH:

END OF FILE
B
-COPY:/ /,/CITY/
```

(The space character between the first set of delimiters causes the blank line at the beginning of the file to be included with the copied text.)

```
-PASTE:/DEAR/;*

END OF FILE - REQUEST EXECUTED   2 TIMES
B
-FIND:/MR./
-PRINT;*
MR.  A.  A. ADAMS
ADDRESS
CITY,STATE

DEAR MR. ADAMS:

WE TAKE GREAT PLEASURE IN ANNOUNCING
                 .
                 .
                 .
                 .
YOURS VERY TRULY,

COMPANY NAME
ADDRESS
CITY,STATE
```

.
.
.
DEAR MR. ZILCH:

WE TAKE GREAT PLEASURE IN ANNOUNCING
.
.
.
.
YOURS VER\ TRULY,

COMPANY NAME
ADDRESS
CITY,STATE

END OF FILE

Two special forms of the operand are permissible with the PASTE command to identify lines containing specified strings. These forms of the command are referred to as the Boolean AND and OR functions. The operand can consist of up to five strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the paste. For example, with d representing a delimiter, the format is

PASTE:dSTRING1d+....+dSTRING5d

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the paste. For example, with d representing a delimiter, the format is

PASTE:dSTRING1d-...-dSTRING5d

Note that these two special forms of the operand are equivalent in line or string mode.

PRINT Command

The PRINT command is used when either a selected portion of a file or the entire file is to be printed. The user can vary the PRINT command to print any one of the following:

- o The entire file
- o Any number of consecutive lines
- o Any number of lines containing a given character string or strings
- o From one point to another
- o A single line

The formats and execution are as follows:

Command	Execution
PRINT	Print one line.
PRINT;n	Print n consecutive lines.
PRINT-j;n	Backup the line pointer j lines and print n lines.
PRINT+j;n	Move the line pointer forward j lines and print n lines.
PRINT;*	Print file from present location of search pointer to end-of-file.
PRINT:/xxx/	Print the line identified by xxx.
PRINT:/xxx/;n	Print the next n lines identified by xxx. (* can be used instead of n to print all such lines.)
PRINT:/xxx/,/yyy/	Print the block of lines starting with the line identified by xxx through the line identified by yyy. (A repeat field can be used to print n or all such lines.)
PRINTS:/yyy/	Print the line containing the specified string.
PRINTS:/yyy/;n	Print n lines containing the specified string. * can be used to print all lines containing the specified string. If the string occurs more than once in a line, the line is printed for each occurrence of the string.
PRINTS:/yyy/,/zzz/	Print from the line containing string yyy to the line containing string zzz, inclusive. (A repeat field can be used with this form also.)
PRINT:/xxx/+/yyy/+...	Print the line containing all of the specified (a maximum of five) strings. (A repeat field can be used to print n or all such lines.)
PRINT:/xxx/-/yyy/-...	Print the line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to print n or all such lines.)

To print the complete file, use the PRINT command in line mode with the asterisk in the repeat field. Printing begins at the location of the search pointer and continues to the end of the file.

```
 -PRINT;*
<>PROGRAMMING LANGUAGES

 HUMAN LANGUAGES ARE IMPRACTICAL FOR PREPARING
 COMPUTER PROGRAMS BECAUSE THESE LANGUAGES
 CONTAIN MANY AMBIGUITIES AND REDUNDANCIES;
 THE COMPUTER INTERPRETS LANGUAGE ABSOLUTELY
 LITERALLY. BY THE SAME TOKEN, MACHINE
 LANGUAGES ARE ALSO IMPRACTICAL BECAUSE THEY ARE
 DIFFICULT FOR PEOPLE TO USE. MOST PROGRAMMING
 LANGUAGES ARE COMPROMISES BETWEEN HUMAN AND
 MACHINE LANGUAGES.

>END OF FILE
```

To print a single line, use the PRINT command in line mode, with or without a string field. If no string field is specified, the line where the search pointer is located is printed.

```
-PRINT
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
-
```

When a string field is specified, the line identified by the string is printed. The string field must contain characters unique to the beginning of the line and only one string field can be used.

```
-BACKUP
-PRINT:/HUMAN/
HUMAN LANGUAGES ARE IMPRACTICAL FOR PREPARING
-
```

To print any number of consecutive lines, use PRINT in the line mode with a repeat field. Printing begins at the location of the search pointer.

```
  -BACKUP
  -PRINT;3
<>COMPUTER PROGRAMS

  >A COMPUTER PROGRAM IS A SET OF INSTRUCTION THAT
   -

   Line space inserted during build
```

To print a specified string, use PRINTS with a string field and with or without a repeat field.

```
-PRINTS:/SHAR/
TIME-SHARING SYSTEM
-PRINTS:/SHAR/;4
TIME-SHARING SYSTEM
THE TIME-SHARING SYSTEM USES A TECHNIQUE BY
THUS, TIME-SHARING PERMITS A USER TO WORK
MANY OTHERS AT THE SAME TIME SHARE THIS
-
```

To print from point-to-point, use PRINTS and two string fields.

```
  -PRINTS:/TIME/,/USE./
<>TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
  COMPUTER AND USER, PERMITTING THE DIALOGUE
  TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
  THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
  DATA IS FED FROM THE TERMINAL DIRECTLY TO
  THE COMPUTER AND ANSWERS ARE RECEIVED
  QUICKLY AT THE SAME TERMINAL.

  IF THE PROGRAM CONTAINS A MISTAKE, THE
  COMPUTER INFORMS THE USER.

  THE PROGRAM CAN BE CORRECTED OR CHANGED BY
  THE USER AS IF HE WERE CONVERSING BY PHONE,
  EXCEPT IN THIS CASE, THE CONVERSATION IS
  TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
 >OF TERMINAL IN USE.
  -
```

The first string field must contain data unique to the first line printed and the second string field must be unique to the last line printed. In the above example, if the second string field did not contain the period after USE, only the two lines of text, through the line containing the word USER, would have been printed.

Two special forms of the operand are permissible with the PRINT command to identify lines containing specified strings. These forms of the operand are referred to as Boolean AND and OR functions. The operand can consist of up to five separate strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the print. For example, with d representing a delimiter the format is

```
PRINT:dSTRING1d+....+dSTRING5d
```

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the print. For example, with d representing a delimiter, the format is

```
PRINT:dSTRING1d-...-dSTRING5d
```

Note that these two special forms of the operand are equivalent in line or string mode.

```
-PRINT;*
A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK. EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
AS DIRECTED BY THE PROGRAM.

A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
BEFORE IT CAN BE RUN (HAVE ALL INSTRUCTIONS
EXECUTED) ON A COMPUTER.

     THE PROGRAM MUST BE SUBMITTED TO THE
     COMPUTER IN A LANGUAGE THAT THE
     COMPUTER RECOGNIZES.

     ALL LANGUAGE INSTRUCTIONS MUST BE
     COMPLETE AND BE PRECISELY STATED.

END OF FILE
BACKUP

-PRINT:/COMPUTER/+/PRODUCES/
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
-BACKUP
-FIND:/TWO/-/BEFORE/-/RECOGNIZES/
-PRINT
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
```

REPLACE Command

The REPLACE command allows the user to replace any number of characters, words, or lines of text with new text of any length. REPLACE may or may not have an operand field. If no operand field is given, the line where the search pointer is located is replaced.

The operand fie d can take one of two forms, depending on the length of the replacement text. Th: first four of the following formats illustrate the format to be used when the operand field cannot be contained in one line. The remaining formats illustrate the use of REPLACE with short strings.

The formats and execution are as follows:

Command	Execution
REPLACE	Replace the line at which search pointer is currently located. (A repeat field can be used with this form.)
REPLACE:/xxx/	Replace the line identified by xxx.
REPLACE:/xxx/;n	Replace the next n lines identified by xxx. (* can be used instead of n to replace all such lines.)
REPLACE:/xxx/,/yyy/	Replace the block of lines starting with the line identified by xxx through the line identified by yyy.
REPLACES:/yyy/	Replace the specified string.
REPLACES:/yyy/;n	Replace n successive occurrences of the specified string. (* can be used instead of n to replace all such occurrences.)
REPLACES:/yyy/,/zzz/	Replace text between points yyy and zzz, inclusive. (A repeat field can be used with this form also.)
REPLACE:/stl/+.../stn/	Replace the line containing all of the specified (a maximum of five) strings. (A repeat field can be used to replace n or all such lines.)
REPLACE:/stl/-.../stn/	Replace the line containing any one of the specified (a maximum of five) strings. (A repeat field can be used to replace n or all such lines.)

Following the REPLACE commands above, the system responds with ENTER. The replacement text is then typed in. Following the ENTER response, the replacement text must include all desired blanks and carriage returns. Replacement text is typed on lines following ENTER. When text entry is complete, a carriage return in response to the asterisk generates the - response.

In string mode, the carriage return on the last line of text is ignored. When replacing short strings of text, the formats shown below can be used.

NOTE: The command and the entire operand field must be on the same line. This format does not accept a carriage return before the final delimiter. The ENTER response is not given with this use of the command.

Command	Execution
REPLACE:/xxx/:/bbb/	Replace line identified by xxx with the string (line) bbb.
REPLACE:/xxx/;n:/bbb/	Replace the next n lines identified by xxx with the string (line) bbb. (* can be used instead of n to replace all such lines.)
REPLACES:/yyy/;n:/bbb/	Replace n successive occurrences of the string yyy with string bbb. (* can be used instead of n to replace all such occurrences.)
REPLACES:/yyy/,/zzz/:/bbb/	Replace text between points yyy and zzz, inclusive, with string bbb. (A repeat field can be used with this form also.)

To replace a string of characters, use REPLACE in the string mode with a string field and with or without a repeat field. Replacement begins at the first character position specified in the operand string field. If a repeat field is specified, n identical replacements are performed (unless end-of-file is encountered first).

```
-PRINT
A PROGRAM MUST MEET TWO PRIMERY REQUIREMENTS
-REPLACES:/ERY/:/ARY/
-PRINT
A PROGRAM MUST MEET TWO PRIMARY REQUIREMENTS
-
```

To replace a complete line, use REPLACE in the line mode, with or without a string field and/or repeat field. The string field, when used, must contain the characters unique to the beginning of the line. When no string or repeat field is given, the line where the search pointer is located is replaced.

Example 1

```
-PRINT
TIME-SHARING LANGUAGES
-REPLACE:/TI/:/TIME-SHARING SYSTEM/
-PRINT
TIME-SHARING SYSTEM
-
```

Example 2

```
-PRINT
TIME-SHARING LANGUAGES
-REPLACE
ENTER
*TIME-SHARING SYSTEM
*(carriage return)
-PRINT
TIME-SHARING SYSTEM
-
```

When the repeat field is used, the lines beginning with the specified string are replaced the indicated number of times. If no string field is given, the indicated number of lines is replaced.

```
-PRINT;14
THE TIME-SHARING SYSTEM USES A TECHNIQUE BY
WHICH PROGRAMS ARE HANDLED IN PARALLEL.  A
SUPERVISORY PROGRAM ACTS AS A CONTROLLER OF
THESE PROGRAMS, CONTROLLING "STOP" AND "GO"
SIGNALS TO INPUTS FROM TERMINALS AND
PREVENTING DEMANDS OF ONE TERMINAL FROM
INTERFERING WITH DEMANDS OF OTHER TERMINALS.
THUS, TIME-SHARING PERMITS A USER TO WORK
DIRECTLY WITH THE COMPUTER, WHETHER IT IS
WITHIN HIS SIGHT OR THOUSANDS OF MILES
AWAY.  THE USER BELIEVES THAT HE HAS
EXCLUSIVE USE OF THE COMPUTER, EVEN THOUGH
MANY OTHERS AT THE SAME TIME SHARE THIS
ILLUSION.
-BACKUP;13
-REPLACE;2
ENTER
*A TIME-SHARING SYSTEM
*(carriage return)
-PRINT;5
A TIME-SHARING SYSTEM
SUPERVISORY PROGRAM ACTS AS A CONTROLLER OF
THESE PROGRAMS, CONTROLLING "STOP" AND "GO"
SIGNALS TO INPUTS FROM TERMINALS AND
PREVENTING DEMANDS OF ONE TERMINAL FROM
```

To replace from point-to-point, use REPLACE in the string mode with two string fields. A repeat field can be used if desired.

```
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER, PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE  SAME TERMINAL.
```

```
THE  PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE OF
TERMINAL IN USE.

END OF FILE
B;13
-REPLACES:/SAME/,/THE/
ENTER
*SAME TERMINAL.
*(blank,carriage return)
*IF THE PROGRAM CONTAINS A MISTAKE, THE
*COMPUTER INFORMS THE USER.
*(blank,carriage return)
*THE
*(carriage return)
-B;11
-PRINT;*
TIME-SHARING PERMITS A DIALOGUE BETWEEN THE
COMPUTER AND USER PERMITTING THE DIALOGUE
TO BEGIN IMMEDIATELY, WITHOUT WAITING FOR
THE COMPUTER TO COMPLETE PREVIOUS PROGRAMS.
DATA IS FED FROM THE TERMINAL DIRECTLY TO
THE COMPUTER AND ANSWERS ARE RECEIVED
QUICKLY AT THE  SAME TERMINAL.

IF THE PROGRAM CONTAINS A MISTAKE, THE
COMPUTER INFORMS THE USER.

THE  PROGRAM CAN BE CORRECTED OR CHANGED BY
THE USER AS IF HE WERE CONVERSING BY PHONE,
EXCEPT IN THIS CASE, THE CONVERSATION IS
TYPED OR DISPLAYED, DEPENDENT UPON THE TYPE
OF TERMINAL IN USE.

END OF FILE
```

Two special forms of the operand are permissible with the REPLACE command to identify lines containing specified strings. These forms of the command are referred to as the Boolean AND and OR functions. The operand can consist of up to five strings connected by plus signs for the AND form and minus signs for the OR form. The strings can be in any order; i.e., the fifth string in order of appearance in the line can be listed first in the operand.

For the AND form, the user lists strings and plus signs to imply that the form is a Boolean AND--all of the strings listed must be present to achieve the replace. For example, with d representing a delimiter, the format is

```
REPLACE:dSTRING1d+...+dSTRING5d
```

For the OR form, the user lists strings and minus signs to imply that the form is a Boolean OR--one of the listed strings must be present to achieve the replace. For example, with d representing a delimiter, the format is

```
REPLACE:dSTRING1d-...-dSTRING5d
```

Note that these two special forms of the operand are equivalent in line or string mode.

RUNOFF Command

The RUNOFF command enables the user to access the RUNOFF subsystem from the EDITOR subsystem. When the user gives the RUNOFF command, the RUNOFF subsystem generates the "ready" response to indicate its availability to accept a RUNOFF subsystem command. After the user has performed desired RUNOFF functions, a DONE command re-accesses the EDITOR subsystem.

STRING Command

The STRING command causes the commands which follow to be executed in the string mode. It is equivalent to adding the S mode indicator to each command typed.

NOTE: Since the first four characters of STRING and STRIP are equivalent, the system command STRIP does not function from within the Text Editor; i.e., the string mode is set instead.

STRING never takes an operand field; however, if the commands which follow STRING do not have a string field included, they operate as if in the line mode.

```
-STRING
-PRINT;6
A COMPUTER PROGRAM IS A SET OF INSTRUCTIONS THAT
TELLS A COMPUTER HOW TO ACCOMPLISH A SPECIFIC
TASK. EACH INSTRUCTION IS PERFORMED IN THE
SEQUENCE SPECIFIED BY THE PROGRAM. IN THIS WAY,
THE COMPUTER PROCESSES AND PRODUCES INFORMATION
AS DIRECTED BY THE PROGRAM.
-BACKUP
-REPLACE:/TASK/
ENTER
*JOB
*(carriage return)
-PRINT:/JOB/
JOB. EACH INSTRUCTION IS PERFORMED IN THE
-

Line mode action
String mode action
String mode action
```

TRANSPARENT Command

The TRANSPARENT (T) command allows the user to search the current file for all transparent characters (nonprinting characters octal 000 through 037). The search begins at the current line pointer position through to the end of file. Each line found containing transparent characters is printed and the character is bracketed by asterisks and printed in translated form. For example, a line containing a backspace would be printed as follows:

```
-T

This line has a backspace *BSP* here.
```

VERIFY Command And NOVERIFY Command

The VERIFY command enables the user to set the mode of the EDITOR subsystem so as to verify the execution of an EDITOR command. For positioning commands, the VERIFY command causes a printout of the line at which the search pointer is positioned when the positioning command is finished. For text altering commands in line mode, the VERIFY command causes a printout of the line preceding the change, the affected change, and the line following the change. Although the line following the change is printed, the search pointer remains at the last line affected.

For text altering commands in string mode, the VERIFY command causes a printout of the one or more lines affected by the change. The NOVERIFY command removes the VERIFY mode.

```
*EDITOR OLD filename
-VERIFY
(EDITOR positioning and text altering commands are verified
by the EDITOR subsystem upon execution. The VERIFY command
will remain in effect until nullified by a NOVERIFY command.)
```

Verification of a particular EDITOR command is achieved by appending the letter V to the command verb.

```
-FINDV:/xxx/;n

(Upon finding the nth occurrence of the specified line,
the line is printed out.)

-REPLACEVS:/xxx/:/bbb/

(Upon replacement of string xxx by string bbb, the altered line
is printed out.)
```

The appended V affects the command once only; the verification is not repeated for subsequent uses of the command.

WHERE Function

The WHERE function provides the user with the current internal block number and the location of the current line within the block. For example:

```
-WHERE          (User types "WHERE" - short form "W" is acceptable)

OCTL BLK#xxxx   (Text Editor identifies the current block number)

RCW=nnn         (Text Editor identifies the address of the current line)
```

Where: xxxx is the current block number (octal) and nnn is the address of the current line RCW (record control word) within the current block

Usage is principally technical, where a user desires to interrogate octal data within a file, or to patch data within a file. The octal block number cannot exceed 7777; otherwise the count will roll over, providing a false block number.

RUNOFF SUBSYSTEM

The RUNOFF subsystem allows the user to print a text file in a previously determined format. The format is directed by control words entered in the file. The RUNOFF control words can be entered during building of the file or inserted later during editing of the file.

In addition to imbedded control words, RUNOFF also uses commands that control the way in which the file is to be saved or printed. These commands are used after entering RUNOFF and are never inserted in the file.

RUNOFF COMMANDS

The RUNOFF subsystem permits the use of the following commands:

EDITOR
NOSTOP
NUMBER
PRINT
REFORM
SKIP n

EDITOR Command

The EDITOR command can be used to access the EDITOR subsystem while in RUNOFF subsystem without the need to return to the subsystem selection level. Upon being given the EDITOR command, the EDITOR subsystem responds with the "-" response. The user can then perform desired editing function and return to the RUNOFF subsystem by means of the Time Sharing System command DONE.

A current file must have been created if the EDITOR command is to be used while in the RUNOFF subsystem. If the system selected at logon time is RUNOFF and no current file exists, the use of the EDITOR command generates the message

```
<52>  CURRENT FILE NOT DEFINED
```

NOSTOP Command

The NOSTOP command can be used when the terminal is loaded with continuous paper. RUNOFF does not stop after each page is printed. The SKIP _n_ command can be used with NOSTOP. The form NOSTOP _n_ permits _n_ consecutive pages to be printed before a stop is made at the end of the _n_th page.

```
PRINT filename2
READY
SKIP 8
READY
NOSTOP
READY
(carriage return)
POSITION PAPER NOW
(carriage return)
(Printing begins at the ninth page and continues to the end
of the file unless stopped manually at the terminal.)
```

NUMBER Command

The NUMBER command indicates the user has a line-numbered file and desires to reformat the file without line numbers. The usage of the command is the same as for the SKIP and NOSTOP commands.

PRINT Command

The operand of the PRINT command contains only one field--the file name of a previously formatted text file. The file name must be the same as the second field of the REFORM command which saved the text.

```
PRINT filename2
READY
```

After a REFORM or PRINT command, the system types out READY. The file(s) specified are accessed but are yet to be acted upon. At this time, the commands SKIP n and NOSTOP n can be entered.

If printing is to be done, READY may be followed by a carriage return, or one or both of two commands--SKIP n and NOSTOP n.

If only a carriage return is used, the formatted text is printed out one page at a time, beginning at the first page of the file. After each page is complete, RUNOFF stops to allow the paper for the next page to be placed in the terminal. When the new paper is positioned, type a carriage return to start printing again. When all pages have been printed, RUNOFF COMPLETE is typed out.

```
REFORM filename1,,PRINT
READY
(carriage return)
POSITION PAPER NOW
(carriage return)
```

REFORM Command

The operand of the REFORM command can contain four fields, separated by commas, and must contain at least two fields.

The first field specifies the file name of the data to be formatted. This field is required. If the file is a current file accessed by another time sharing subsystem, an asterisk can be used in the first field in lieu of the file name.

The second field specifies the file name under which the formatted data is to be saved. This field is optional, but must be present when the third field is not used.

The third field contains the command PRINT. This field is optional, but must be present when the second field is not used.

The fourth field contains the expression COUNT n. COUNT produces, in formatted text, the relative line number of the source file specified in the first field. n indicates the number of spaces set for the left margin of the formatted text. This fourth field is optional. Where the field PRINT is not used, the COUNT n field can replace it. The order of the fields can not be changed. The n portion of COUNT n is optional, with the following actions resulting:

1. If n is not present and RUNOFF does not encounter a left margin, six spaces are provided for the left margin.

2. If n is not present and RUNOFF does encounter a left margin setting, this margin setting is used.

3. If n is present and RUNOFF does not encounter a left margin setting, n designates the setting.

4. If n is present and RUNOFF does encounter a left margin setting, the setting is the larger of the two.

The following examples illustrate the use of the command (COUNT n can be used with any of the examples).

```
REFORM filename1,filename2,PRINT
READY
```

This form of the command causes file 1 to be formatted into pages and saved in file 2. At the same time, the formatted contents of file 2 are printed out at the terminal.

```
REFORM filename1,filename2
READY
```

This command formats file 1 into pages and saves the formatted text in file 2, to be printed out at a later time. (File 2 must be a previously defined file.)

```
REFORM filename1,,PRINT
READY
```

This command formats file 1 and transmits the formatted text to the terminal. (The contents of file 1 saved by EDITOR remain saved in unformatted form.)

SKIP n Command

The SKIP n command allows the user to obtain partial output of the file. Printing begins at page n+1. When printing stops at the end of each page, this command can be used.

```
PRINT filename2
READY
SKIP 8
READY
(carriage return)
POSITION PAPER NOW
(carriage return)
(The ninth page is printed out)
SKIP 3
READY
(carriage return)
(The thirteenth page is printed out)
```

RUNOFF FORMAT CONTROL WORDS

The RUNOFF format control words which can be entered in the text file during the building or editing process are listed below. Each of these can be used in an abbreviated form, utilizing the first four letters (e.g., .allc).

.allcaps n
.beginpage n
.boldface n
.bottommargin n
.break
.center n
.comment
.doublespace
.fill
.footing x,n
.header x,n
.ignore x,x
.indent n
.justify
.leftdent n
.linelength n
.literal
.margin t,b,l,r
.multispace n
.nodent
.nofill
.nojust
.notab
.page x,y,n
.paperlength n
.paragraph
.point n
.reference (x...x)
.replace x,x
.scoreunder n
.singlespace
.space n
.subheading x,n
.subfooting x,n
.subparagraph n
.tabulate t,n,,,n
.topmargin
.undent n

The following rules apply to use of RUNOFF format control words:

1. Each control word must be preceded by a period and followed by a carriage return. Any text material typed on the same line as the control word is ignored when printing out the formatted text.

2. Control words can be typed in either uppercase or lowercase.

3. All legitimate control words are ignored when printing out and do not appear in the text.

4. Control words that are to remain in effect throughout the file can be entered once at the beginning of the file and need not be repeated unless they are cancelled by an imbedded control word. For example:

```
.PAPE 66
.LINE 60
.TOPM 6
.BOTT 6
.SING
.FILL
.JUST
```

5. The control words and values shown in the above example are those preset by RUNOFF and need not be entered; they remain in effect unless changed by the user. No page numbering occurs unless .PAGE is encountered. Care should be exercised in specifying page size parameters. RUNOFF formats a full page before printing. The following page matrix formula can be used to determine a large page format. Exceeding the results of this calculation leads to a memory fault.

```
4P + (P-T-B+2)(L+2) = 7000
where P = .paperlength n
      T = .topmargin n
      B = .bottommargin n
      L = .linelength n
```

6. Words should not be hyphenated at the end of a line when using .FILL. The carriage return following the hyphen is treated as a space character and the hyphenated word could appear in the middle of a line of text as follows:

```
MULTI- PLIED
```

A compound, such as "right-hand", is treated as one word by RUNOFF and is not split over two lines in order to fill or justify lines.

.ALLCAPS n

Print next n lines in uppercase. If n is not used, only the next line is printed in uppercase.

.BEGINPAGE n

Place text following control word on a new page. If n is specified, the new page is numbered n and succeeding pages are referenced by n.

.BOLDFACE n

Overprint the next n lines. If n is not used, only the next line is overprinted. The use of .BOLDFACE and .SCOREUNDER on the same line(s) results in .SCOREUNDER operation only.

.BOTTOMMARGIN n

Specify the space from the last line of text output to the bottom of the paper. n should equal the number of lines desired. If this control word is not used, RUNOFF presets the margin to 6. Page numbers, if requested, are printed within the margin space.

.BREAK

End previous line and start a new line, without inserting a blank line. The lines previous and following the use of this control word are not joined even though .FILL has been specified.

.CENTER n

Center the next n lines. When n is not used, only the next line is centered. When centering, do not include any other RUNOFF control words within the lines to be centered.

.COMMENT

Prevent printing of all lines of text until another RUNOFF control word is encountered.

.DOUBLESPACE

Specify text to be printed out double spaced.

.FILL

Lengthen short lines by moving words from the following line and shorten long lines by moving words to the following line. This is preset by RUNOFF and is in effect until a .NOFIL is encountered. .FILL does not insert spaces to justify the right-hand margin.

.FOOTING x,n

Specify the number of lines and the position of the foot line to be printed at the bottom of a page. One line space is automatically inserted before the footing.

n indicates the number of lines. X can be one of the following:

C - Centered on each page.
R - Right justified on each page.
L - Left justified on each page.
A - Alternately right justified on odd numbered pages,
    left justified on even numbered pages.
E - Left justified on even numbered pages.
O - Right justified on odd numbered pages.

The .FOOTING control word can be entered only at the beginning of the file or after .BEGINPAGE within the file if the foot line is to be changed. Termination of foot lines is accomplished by use of .FOOTING NO or .FOOTING 0 (numeric).

.HEADER x,n

Specify the number of lines and the position of the header to be printed on a page. One line space is automatically inserted after the header. To insert a blank line in the header, use a space character before the carriage return.

N indicates the number of lines. X can be one of the following:

C - Centered on each page.
R - Right justified on each page.
L - Left justified on each page.
A - Alternately right justified on odd numbered pages, left justified on even numbered pages.
E - Left justified on even numbered pages.
O - Right justified on odd numbered pages.

For example:

```
*.HEADER R,3
*TIME-SHARING
*(space)
*(space)
*
```

The .HEADER control word can be entered at the beginning of the file and also just before or after .BEGINPAGE within the file if the heading is to be changed. Termination of header lines is accomplished by use of .HEADER NO or .HEADER 0 (numeric).

.IGNORE x,x,.....

Prevent the symbols listed in the operand from being used as text characters. Up to 16 characters may be listed for suppression. Use of the characters as text is resumed by .IGNORE NO or .IGNORE 0. Numerics are not valid symbols for use with .IGNORE.

.INDENT n

Indent each following line of text the number of spaces specified. Indentation is preset to zero and is cumulative; that is, subsequent .INDENT control words add to the total indentation until a .NODENT, .LEFTDENT, or .UNDENT is encountered.

.JUSTIFY

Insert spaces into the line between words to justify the right-hand margin to the length specified by .LINE. This is preset by RUNOFF and remains in effect until a .NOJUST is encountered.

.LEFTDENT n

In an indented area, subtract n spaces from the total indentation, for all following lines until an .INDENT, .NODENT, or .UNDENT is encountered. If n is greater than the total indentation, the total is set to zero.

.LINELENGTH n

Specify the length of the line, in characters, for filling and justifying. N should equal the length in inches multiplied by 10. (6-inch line = 60, 7-inch line = 70, etc.) If this control word is not used, RUNOFF presets the line to 60. The left margin position on the paper is determined manually at the terminal.

.LITERAL

Print a RUNOFF control word when it appears as part of the text. .LITE can be used on the same line, preceding the control word, or on the line before the control word as shown below.

```
.LITERAL
.LITE can be used on the same line,
.LITERAL .PAGE n starts page numbering.
```

.MARGIN t,b,l,r

Set the four margins of a page. The numerics for T(top) and B(bottom) set the line count for the top and bottom margins. Numerics for L(left) and R(right) set the character counts for left and right margins. T (top) and B (bottom) margins must be specified. Nulling of relative fields will result in a top and bottom margin of zero.

NOTE: The .MARGIN control word cannot be utilized to change top or bottom margins in the middle of a page. To change top or bottom margins on a succeeding page, use .MARGIN within the bounds of the page, immediately following .BEGINPAGE or the page break.

.MULTISPACE n

Specify text is to be printed out with n line spaces between text lines. This command overrides any .SINGLESPACE or .DOUBLESPACE command.

.NODENT

In an indented area, reset the total indentation to zero.

.NOFILL

Print all lines exactly as they were typed into the file.

.NOJUST

Stop justification.

.NOTAB

Stop tabulatior and return to the previous format instructions.

.PAGE x,y,n

Start page numbering. If n is not present, numbering begins with page 1. If page numbers are to start with any other number, n should equal the starting page number.

X and y specify where page numbers are to appear. X and y can take one or more of the following values:

B - Bottom of page
T - Top of page
C - Center
L - Left-justified
R - Right-justified
A - Alternating (odd numbers on the right, even on the left)

Page numbers, if requested, are inserted within the specified margin.

The example below would cause numbering to begin with page 1, numbers to be printed on alternate sides of the pages, at the bottom.

```
.PAGE B,A,1
```

If n is specified and x and y are not, page numbers appear centered and at the bottom of the page.

.PAPERLENGTH n

Specify the total length of the paper. n should equal the length in inches multiplied by 6. (11-inch paper = 66, 14-inch paper = 84, etc.) If this control word is not used, RUNOFF presets the length to 66.

.PARAGRAPH

Preset the line length to its specification before the previous .SUBP control word. In an indented region, the former indentation total regains control.

.PARAGRAPH n1,n2

The n1 field causes the left margin to be indented the number of spaces specified. The n2 field causes the first line of the paragraph to be indented the number of spaces specified.

.POINT n

Cause a new page to be formatted. The page number is not incremented, but appears with a period followed by 1 (p.1). The page incrementing continues behind the period until terminated with the control word .BEGIN, when page p resumes incrementing. If the operand n is used, the 1 following the period is replaced with n.

.REFERENCE (x...x)

This causes the text within the parentheses to be printed as a footnote at the bottom of the same page. The .REFE must be preceded by a footnote indicator that must also be the first character(s) within the parentheses; i.e., no space is permitted between the first parenthesis and the indicator.

```
- - - - - - - - - - - - - - - - - - - - -          see
(1)
.REFE ((1) This text is a footnote  printed  at  the
bottom of the page.) below.- - - - - - - - - - - - -
-
```

When printed by RUNOFF, appears as

```
- - - - - - - - - - - - - - - - - - - - - see (1)
below.- - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - -   -
- - - - - - - - - - - - - - - - - - - - - - -   -
- - - - - - - - - - - -

(1) This text is a footnote printed at the bottom
of the page.
```

.REPLACE x,x.....

Cause the symbols listed in the operand to be replaced with space characters. The space characters supplied are not used as word string terminators in formatting the text. This enables the user to reserve character spaces for special character insertion, superscripting, subscripting, etc. Up to 16 symbols can be listed for replacement. Use of the symbols as text is resumed by the control word .REPLACE NO. Numerics are not permitted as REPLACE characters, such usage may cause unpredictable results.

.SCOREUNDER n

Cause the next input text line or each of the next n input text lines to be underscored (underlined) in the formatted text. If n is omitted, underscoring is performed only on the next line of text. The use of .BOLDFACE and .SCOREUNDER on the same line(s) results in .SCOREUNDER operation only.

.SINGLESPACE

Specify text to be printed out single spaced. If this control word is not used, and if .DOUBLESPACE is not specified, RUNOFF presets the format to single space.

.SPACE n

Insert n blank lines spaces. If the end of the page is reached before n (spaces) are provided, spacing stops. Blank lines are not carried over to the next page.

.SUBHEADING x,n

Specify the number of lines to be printed as a subheading to a previously defined header.

N indicates the number of lines. X can be one of the following:

C - Centered on each page.
R - Right justified on each page.
L - Left justified on each page.
A - Alternately justified on odd-numbered pages,
    left justified on even-numbered pages.
E - Left justified on even-numbered pages.
O - Right justified on odd-numbered pages.

The .SUBHEADING control word can be changed after a .BEGINPAGE within the file. Termination of the use of the subheading is accomplished using .SUBHEADING NO or .SUBHEADING 0 (numeric).

.SUBFOOTING x,n

Specify the number of lines to be printed as a subfooting to a footing previously defined.

N indicates the number of lines. X can be one of the following:

C - Centered on each page.
R - Right-justified on each page.
L - Left-justified on each page.
A - Alternately right justified on odd-numbered pages,
    left-justified on even-numbered pages.
E - Left-justified on even-numbered pages.
O - Right-justified on odd-numbered pages.

The .SUBFOOTING control word can be entered only at the beginning of the file or after a .BEGINPAGE within the file. Termination of the use of the subfooting is accomplished using .SUBFOOTING NO or .SUBFOOTING 0 (numeric).

.SUBPARAGRAPH n

Indent the beginning of each line n and subtract n spaces from the end of the line. For example, if the line length is 60 and .SUBP 5 is used, the lines following are 50 characters long.

In an indented region, the subparagraphing is affected by the total indentation. For example:

```
.LINE 60
.INDENT 5
.SUBP 5
```

results in lines 45 characters long, indented 10 spaces from the left margin.

.TABULATE t,n,...

Set simulated tabs on the horizontal line locations specified by the values of n. When building the file, enter a tabulation character (any keyboard character other than a blank, control character, or one being used as a delimiter) at the beginning of each tabulated field, as this character is used by RUNOFF. See the following examples.

```
.TABU  t,10,20,30
txxxxtyyyytzzzz
txxxxtyyyytzzzz
```

When printing in RUNOFF, the following results:

```
(columns) 10           20           30
           xxxx         yyyy         zzzz
           xxxx         yyyy         zzzz
```

When using a terminal that has no tab control key, any symbol can be chosen as a tabulation character. The symbol is not printed out during RUNOFF but can be read when using EDITOR. .TABU operates in a .NOFIL environment.

.TOPMARGIN n

Specify the space from the top of the paper to the first line of output. N should equal the space desired multiplied by 6. (1-inch margin = 6, etc.) If this control word is not used, RUNOFF presets the margin to 6. Page numbers, if requested, are printed within the margin space.

.UNDENT n

In an indented area, causes n to be subtracted from the total indentation for the next line only.

RUNOFF EXAMPLES

Examples are given on the following pages to illustrate the use of RUNOFF. The left-hand page contains the text and instructions in the file. The right-hand page shows the same portion of the file as it is formatted by RUNOFF.

```
.pape 65
.line 67
.page 1
.topm 6
.bott 6
.just
.repl &
.header r,l
Text Editor
.subheading r,l
Examples
.space 4
.cent
SECTION I
.space 2
.cent
INTRODUCTION
.space 4
     This manual describes the Text-Editing Subsystems of the
Time Sharing System, EDITOR and RUNOFF. Use of these subsystems
does not require any knowledge of programming; however, the
following brief descriptions of computer systems and the terms
used in the manual will be helpful to the terminal operator.
.space 2
     In this manual, a "computer system" is an information
processing system. It may be located many miles from the
terminal through which information is being entered. The total
system consists of hardware (printers, card readers and punches,
permanent magnetic storage devices, processing equipment, etc.)
and programs (sets of instructions that tell a computer how to
accomplish a specific task). The Time Sharing System is one of
many such programs.
.space 2
     The Time Sharing System is made up of several small
programs called "subsystems". (See Figure 2 and (1) below.)
This manual covers two of these subsystems in detail. Other
subsystems, not required for text-editing purposes, are covered
in other manuals.
.nojust
.refe ((1) See Text Editor manual, DD18.)
.begi
```

SECTION I

INTRODUCTION

This manual describes the Text-Editing Subsystems of the Time Sharing System, EDITOR and RUNOFF. Use of these subsystems does not require any knowledge of programming; however, the following brief descriptions of computer systems and the terms used in the manual will be helpful to the terminal operator.

In this manual, a "computer system" is an information processing system. It may be located many miles from the terminal through which information is being entered. The total system consists of hardware (printers, card readers and punches, permanent magnetic storage devices, processing equipment, etc.) and programs (sets of instructions that tell a computer how to accomplish a specific task). The Time Sharing System is one of many such programs.

The Time Sharing System is made up of several small programs called "subsystems". (See Figure 2 and (1) below.) This manual covers two of these subsystems in detail. Other subsystems, not required for text-editing purposes, are covered in other manuals.

(1) See Text Editor manual, DD18.

```
.pape 65
.line 67
.repl
.header r,l
Text Editor
.subheading r,l
Examples
.footing c,l
Time Sharing System
.subfooting c,2
Text
Editor
.space 4
.cent
PROCESSOR
.space 3
.nofil
.cent 4
Memory
Where Programs
are Stored
(During Use)
.space 3
.tabu z,10,29,44
zMagnetic Tapes
zDisks, and DrumszInput/OutputzPrinters
zwhere programszControllerszCard Punches
zare stored whenzzCard Readers
znot being used
.notab
.justify
.space 3
.cent
TERMINAL(S)
.space 3
.cent
Figure 1-1. Information Processing System
.space 4
     The following verbs may not have an operand field:
.space 2
.tabu 5,10,31,50
tLINE or LtRUNOFFtSTANDARD
tSTRING or StVERIFY
tBUILDtNOVERIFY
.notab
.fill
.begi
```

```
                    PROCESSOR


                      Memory
                  Where Programs
                    are Stored
                   (During Use)


Magnetic Tapes,
Disks, and Drums    Input/Output    Printers
where programs      Controllers     Card Punches
are stored when                     Card Readers
not being used


                   TERMINAL(S)


       Figure 1-1. Information Processing System
```

The following verbs may not have an operand field:

LINE or L	RUNOFF	STANDARD
STRING or S	VERIFY	
BUILD	NOVERIFY	

```
Time Sharing System
       Text
      Editor
```

```
.pape 65
.line 67
.repl
.header r,l
Text Editor
.subheading r,l
Examples
.space 4
     The use of the verbs and operands are fully explained
and illustrated in
.score
Editor Commands
later in this chapter. The restrictions and usage rules
which apply to the operand field are explained in
.score
Operand Field
below.
.space 2
.subp 5
The editor responds to the commands with messages that
inform the user when a command has been executed, a mistake
in command format has been made, or the end of the file has
been reached. These messages are described in
.score
Responses from Editor.
.para
.space 3
.allcaps
operand field
.space 2
     As stated above, the operand field can contain one
or more of the following:
.space 2
.indent 10
.undent 5
1.   Mode Indicators -
"S" for string mode and "L" for line mode
.space
.undent 5
2.   String field, preceded by a colon
.space
.undent 5
3.   Repeat field, preceded by a semicolon
.leftdent 5
.space 2
If more than one of these items is used in a single
command, the order must be as shown previously.
.nodent
.space 3
.score
Mode Indicators
.space 2
      The mode indicators used with the Editor verbs are
"S" for string mode and "L" for line mode. The mode
determines the type of operation to be performed and the
interpretation of the string field. See Figure 3.
.ignore no
.begin
```

The use of the verbs and operands are fully explained and illustrated in Editor Commands later in this chapter. The restrictions and usage rules which apply to the operand field are explained in Operand Field below.

The editor responds to the commands with messages that inform the user when a command has been executed, a mistake in command format has been made, or the end of the file has been reached. These messages are described in Responses from Editor.

OPERAND FIELD

As stated above, the operand field can contain one or more of the following:

1. Mode Indicators -
   "S" for string mode and "L" for line mode

2. String field, preceded by a colon

3. Repeat field, preceded by a semicolon

If more than one of these items is used in a single command, the order must be as shown previously.

Mode Indicators

The mode indicators used with the Editor verbs are "S" for string mode and "L" for line mode. The mode determines the type of operation to be performed and the interpretation of the string field. See Figure 3.

SECTION XI

Time Sharing System FORTRAN

PROGRAM STATEMENT INPUT

The system is currently in build-mode (as indicated by the initial asterisk) and is ready to accept FORTRAN program statement input or control commands. All lines of input other than control commands are accumulated on the user's current file as they are entered into the system.

Following each line of non-command language input and the terminating carriage return, the system supplies another initial asterisk when the carriage is returned, to indicate the system is ready to accept more input.

Format

A line of FORTRAN input can contain:

1. One or more FORTRAN statements
2. A partial statement
3. A continuation of a statement left incomplete in the preceding line of input
4. A comment
5. A combination of 3 and 1, or 3 and 2, in that order
6. A combination of 1 and 2

A line input must begin with a line-sequence number from one to eight numeric characters. The line-sequence number (line number) enables the programmer to correct and modify the source program.

A line number is distinct from a statement number in that a statement number is a part of the FORTRAN language statement itself.

The line number is always terminated with a single control character that can be a blank, an ampersand, a number sign, an asterisk, or the letter C. This control character merely serves to indicate what type of information follows (i.e., new statement, continuation, or comment) and is not compiled as part of the program. The semicolon can be used to indicate the end of one complete FORTRAN statement and the beginning of another on the same line of input. A carriage return must be used to terminate a complete line of input. This line format is suitable for direct processing by the FORTRAN compiler with the options NFORM and LNO.

The general format of a line of FORTRAN input is

```
nnnnnnnncstatement or continuation ;statement...;statement

                    or

nnnnnnnnc comment
```

Where: <u>nnn...n</u> is a numeric line number, the magnitude of which is less than 2 (262,144)

<u>c</u> is a control character that can be a blank, an ampersand, an asterisk, a number sign, or the letter C, and must immediately follow the last digit of the line number.

<u>Control Character</u>

The control character identifies the type of information that follows it.

b̸ (blank) - If the character position immediately following the last digit of the line number contains a blank, and the next nonblank character is not an ampersand, then that nonblank character is assumed to begin a new FORTRAN statement. In this case, the next nonblank character may begin a FORTRAN statement number (i.e., mm...m statement-text).

(ampersand) - If an ampersand is the first nonblank character following the line number, the next significant character is assumed to be a continuation of the previous statement in the previous line of input. (A blank character is significant only as a continuation of the character string from a preceding line.) The effect of " " is to suppress the previous carriage return as an end-of-statement indicator.

* (asterisk) or C - If the line number is terminated with an asterisk or the letter C, the following information is assumed to be a comment. The comment itself is terminated by a carriage return.

(number sign) - If a numeric character is desired in column 1 of the card image and line numbers exist in the source file, a number sign (#) immediately following the line number causes the character following it to be placed in column 1.

A semicolon within a noncomment line indicates both the end of the preceding statement and the beginning of a new statement. The new statement can include the FORTRAN statement number, mm...m.

The format of a statement that follows a blank control character, is

...nnƀ <u>ƀ...ƀ mm...m FORTRAN-language-text</u>

(The statement format portion is underlined.)

Where: <u>ƀ...ƀ</u> are optional blanks

<u>mm...m</u> is an optional numeric statement number where mm $\leq$ 99999

<u>Blanks (Or Spaces) Within A Line Of Input</u>

Initial, embedded, or trailing blanks in a line of input have no significance in interpretation; however, blanks are illegal within the line number and the nonnumeric character immediately following the line number is interpreted as a control character. Thus, spacing can be used quite freely within a line of input for legibility. Blanks within character constants and nH fields (i.e., alphanumeric information) are meaningful and are retained in the object program coding.

NOTE: The line/statement format is completely free-form, or position independent with the exception of the control character.

To this point, the discussion of line format has been oriented to the NFORM format described earlier in this document. This is generally the most convenient form to use in time sharing, although it is not mandatory. The source file can be built using the Text Editor and can be used without line numbers through the NLNO option. The source program can be in "fixed" format (i.e., without line numbers) through the FORM option. The full spectrum of line formats and source file recording modes is available to the time sharing user.

SOURCE PROGRAM MODIFICATION

Keyboard input is sent to the computer and written onto the user's current file in units of complete lines. A line of terminal input is terminated by a carriage return and no part of the line is transmitted to the system until that carriage return is given. Therefore, corrections or modifications can be done at the terminal at two distinct levels:

1. Correction of a line-in-progress (i.e., a partial line not yet terminated).

2. Correction or modification of the source program (i.e., the contents of the current source file) by the replacement or deletion of current lines, or the insertion of new lines.

The correction of a typing error that is detected before the line is terminated can be done in one of two ways:

- o Delete one or more characters from the end of the partial line

- o Cancel the incomplete line and begin again

NOTE: Use of the delete control character deletes the character preceding the deletion character. (The delete control character used is dependent upon the make of terminal at the site.)

Example:

If # is the deletion character,

```
JONS#
```

deletes S

```
JONS DAVEY#######
```

deletes S DAVEY

Correction or modification of the current source file is done on the basis of line numbers and proceeds accordingly.

Example:

The source file contains

```
100    READ(5,16)HRS,RATE,NO
200    WRITE(6,16)HRS,RATE,NO
300 16 FORMAT(F3.2,F4.2,I6)
```

1. Replacement. A numbered line replaces any identically numbered line that was previously typed or contained on the current file.

Example

```
200    WRITE(6,12)PAY
```

replaces the current line numbered 200.

2. Deletion A "line" consisting of only a line number (i.e., 100) causes the deletion of any identically numbered line that was previously typed or contained on the current file.

Example

```
100
```

deletes line 100 from the source file.

3. Insertion. A line with a line-number value that falls between the line-number values of two pre-existing lines is inserted in the file between those two lines. If the line number is less than the first line number, it is inserted at the beginning of the file; if greater than the largest line number, it is inserted at the end of the file.

Example

```
250 12 FORMAT(//16HPAY IS EQUAL TO ,F6.2)
```

is insert above line 300.

The new source file now contains

```
200    WRITE(6,12)PAY
250 12 FORMAT(//16HPAY IS EQUAL TO ,F6.2)
300 16 FORMAT(F3.2,F4.2,I6)
```

Input Error Recovery

The decimal input/output routine permits the time sharing user (BCD or ASCII) to correct a string of characters in an executing FORTRAN program that was entered from a terminal when a character is illegal for the current format conversion (e.g., a decimal point is illegal in an "I" field). When the current input line is printed on the terminal with a pointer to the illegal character, the correction can be made, and the input/output routine resumes with the new string. If the response is a carriage return, an error message is printed.

At any point in the process of entering file building input in line-numbered subsystems, the LIST command may be given, which results in a clean, up-to-date copy of the current file. In this way, the results of any previous corrections or modifications can be verified visually. Following the command "OLD filename", the LIST command can be used initially to inspect the contents of the current source file (i.e., the "old" program).

I-D-S/II IN A FORTRAN TIME SHARING ENVIRONMENT

The use of I-D-S/II in the FORTRAN time sharing environment requires the ability to specify FORTRAN source files, I-D-S/II control files, and I-D-S/II data base area and key files as well as the desired options from the terminal. The YFORTRAN and FORTRAN time sharing systems provide this capability.

Files Required By I-D-S/II

I-D-S/II requires control files and data base area files. Data base key files and data base procedure files may also be required. The control files required are

- Schema File - the schema file, a random file produced by the schema translation, is the "1*" file unless it has been renamed in the Device Media Control Language (DMCL). It has the alternate name "1.". If 1* has been renamed in the DMCL, it must have that alternate name. The schema file is required in the AFT at execution time.

- Validated Subschema File - The validated subschema file, a random file produced by the subschema translation and validation, has the alternate name "6*" and is required in the AFT at compilation time.

- Subschema Control Structure - Unlike the other I-D-S/II files, the subschema control structure, a sequential file produced by the subschema validation, is not accessed from the AFT. This file, which was referenced by the filecode C* during validation, is bound instead with the FORTRAN object program at load time. It consists of two object modules, S.xxxx and D.xxxx, where xxxx are the first four letters of the subschema name.

Data base area files are required. Data base key files may also be required. Both types of files must be placed in the AFT under their alternate names (i.e., the file codes which were specified in the schema DMCL). The following types of data files can be specified:

- Integrated
- Integrated with Record Keys
- Indexed
- Indexed with Record Keys

If any required data base procedures were not included in the FORTRAN source program itself, files containing these procedures must be supplied. These files, produced during previous compilations, supply the procedures specified in the schema and subschema. These object units, like the control structure, are bound with the FORTRAN object program at load time.

When the DML option is specified, an INVOKE statement in the FORTRAN source program enables the FORTRAN compiler to read the 6* file and obtain the subschema. The subschema then becomes part of the FORTRAN program and defines the User Working Area (UWA).

At run time, the schema file (1*) and the data base area and key files must be in the AFT under the appropriate alternate names. The control structure is used at run time to describe the subset of the data base which is accessible to the program.

Comparison Of The FORTRAN And YFORTRAN Time Sharing Systems

There are two time sharing versions of the FORTRAN compiler. Each version is invoked by the call specified below.

Compiler Version	Language Call
Batch based time sharing compiler	YFORTRAN
Time sharing based compiler	FORTRAN

The time sharing based FORTRAN compiler compiles under the time sharing system rather than being spawned as in the case of the batch based time sharing compiler. It differs from the batch based compiler because it:

- o Compiles under the GCOS time sharing system
- o Eliminates the need for configuring batch memory; YFORTRAN compiles through DRL TASK
- o Significantly reduces overhead in the FORTRAN time sharing system
- o Does not require the "CORE=" clause for compilations
- o Has identical compilers with the exception of the executive phase (YFXC vs YUEX)

THE YFORTRAN TIME SHARING SYSTEM RUN COMMAND

The YFORTRAN time sharing RUN command can be written as either RUN or RUNH. The RUNH form is used to display a heading line on the terminal that gives a date, a time, and a SNUMB. Any of the seven following options can be specified with the RUN (or RUNH) command:

```
RUN [H]  [-nnn]  [fs]  [= fh]  [; fc]  [(opt [,...  )]  [ulib]] [#fe]
```

-nnn — nnn is the maximum processor time (in seconds) the program is allowed to run during execution.

fs — is the set of file descriptors (separated by semicolons) for source files in the standard BCD card image format, in compressed card image format (COMDK), or in time sharing ASCII standard system format, and/or descriptors for binary card image object files. These files serve as inputs to the compiler and/or loader. Concatenation of source files is provided by using a separate semicolon between each file descriptor. Where a BCD or COMDK source file is supplied (media code 1 or 2), fs can also include a descriptor for a BCD alter file. The alter file must begin with a $ UPDATE card and must be in alter number sequence. If there are many BCD or COMDK source files in the list, the alter file updates the first source file. If the FORTRAN program contains I-D-S/II DML statements, fs should also contain the file descriptor for the subschema control structure file. If data base procedures are required and are not supplied as part of the FORTRAN source program, file descriptors for the procedure object files should also be listed here.

Alternatively, the list fs can consist of a single file descriptor that points to a previously generated system loadable (H*) file.

A file descriptor consisting of the single character "*" indicates the current file (*SRC). The fs list is optional, and when missing, indicates that only the current file (*SRC) is to be compiled.

fh — is a single file descriptor of a random file into which the system loadable file (H*) produced by the General Loader is saved if the compilation is successful. This file is written if no fatal errors occur during compilation. If the named file does not exist, a permanent random file of 36 blocks (llinks) is created and added to the user's catalog. If the field is missing, the H* file is generated into a temporary file. The presence of this option is valid only when the program indicated by the list fs, the FORTRAN library, and the user library (if any) is bindable (i.e., no outstanding SYMREFs). If the General Loader indicates that outstanding SYMREFs exist, an executable H* file is created, but any reference to an unsatisfied SYMREF causes the program to be abnormally terminated. (The General Loader inserts a MME GEBORT at references to unsatisfied SYMREFs. When a MME is encountered during the execution of a time sharing subsystem, GCOS and the Time Sharing Executive simulate an illegal operation fault.)

;fc a single file descriptor preceded by a semicolon of a sequential file into which the compiler is to place the binary (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs.

If the named file does not exist, a permanent linked file of three blocks (llinks) is created and added to the user's catalog. This file expands as necessary up to a maximum of 20 blocks (llinks), to hold the object deck(s). In this case, the field fs plus the libraries do not need to indicate a complete program (individual or collections of subroutines can be compiled and saved). When this optional field is missing, a C* file is not generated; when present, the DECK option is activated for the compilation process.

(opt) a list of options available for time sharing which, when specified, must be separated by commas. Some of these options affect the compilation process and some affect the loading process (the default options are underlined).

DEBUG - The run time debug symbol table is generated.

NOTE: This debug symbol table is used for debugging in the batch mode only. Refer to the General Loader manual for use of the debug feature and the debug symbol table.

NDEBUG - The run time debug symbol table is not generated.

BCD - Object character set is BCD. If applicable, this option must be specified whenever the General Loader is to be called. This is required for compile, compile and load, and load activities; it is not required for execute only runs (run H* file). The BCD option cannot be specified if the DML option is selected.

ASCII - Object character set is ASCII.

FORM - Source is in "fixed" format (LNO option is not valid with FORM).

NFORM - Source is in "free" format.

LNO - Source is line-numbered (default option if FORM is not specified).

NLNO - Source is not line-numbered (default option if FORM is specified).

OPTZ - The object module is optimized.

NOPTZ - The object module is not optimized.

NWARN - No compilation warning messages are printed, although, fatal messages are printed,

CORE=nn - The compilation activity memory requirement is set to nnK+9K or 29K, whichever is larger. If not specified, nn is set to 20.

FDS - The FORTRAN Debugging System (FDS) is enabled.

NFDS - The FORTRAN Debugging System is not invoked.

DML - The Data Manipulation Language (DML) facility of I-D-S/II is invoked. If DML is specified, the necessary I-D-S/II files must also be specified in the RUN command. The BCD option cannot be used with the DML option.

STAT - The I-D-S/II statistics are printed. If a sequential file with the alternate name "P." exists in the AFT, the I-D-S/II statistics and abort codes are written to that file. The file is written as a BCD file and can be converted to an ASCII file for examination from a terminal by the command "CONV file descriptor." If "P." does not exist in the AFT, the statistics and abort codes are specified, and written to the terminal. If the STATS option is not specified, the I-D-S/II statistics are not printed and the fatal abort codes are directed to the terminal. A FINISH statement must be included in the FORTRAN program in order to receive any statistics. STAT is valid only when the DML option is specified.

LDEL - Logical record delete is requested. The default is physical record deletion. LDEL is valid only when the DML option is specified.

The remaining options concern the loading process (the default option is underlined).

GO - The program is loaded and executed at the completion of compilation.

NOGO - The program is not executed at the completion of the compilation. If specified, the object program is loaded and saved. If no object (H*) save file is specified, only the compilation is performed (General Loader is not called).

ULIB - File descriptors exist at the end of the options field that allocate user libraries to be searched for missing routines prior to searching for them in the system library.

NOLIB - No user libraries are to be used.

TIME=nnn- The batch compilation and/or General Loader activity time limits are set to nnn seconds; where nnn $\leq$ 180. If not specified, nnn is set to 60.

URGC=nn - The urgency for the batch compilation and/or General Loader activity is set to nn, where nn $\leq$ 40. If not specified, nn is set to 40.

TEST - A test version of the compiler is to be used for the activity. There must be an accessed file (in the AFT) with the name FORTRANY. If these two conditions are met, then file FORTRANY is allocated as file code ** in the activity.

REMO - All temporary files that are created during compilation and loading are removed from the AFT as they are no longer needed. This option keeps the number of files in the AFT down to a minimum but causes more time to be spent processing each RUN command.

NAME=name - Provides a name for the main link of the saved H* file. It can be used at time of creation of this file and subsequently as it is reused. This name is placed in the SAVE/field of the $ OPTION card.

ulib A list of file descriptors (separated by semicolons) pointing to random files containing user libraries to be searched before the system library. This list must be provided by the user when the ULIB option is specified.

#fe A list of file descriptors (the first preceded by a number sign) for files required during execution. Each catalog/file description is separated by a semicolon. The file description can be in any of the following formats:

1. filename in the form filename "nn", represents a logical file code referenced by the I/O statements in the program where 01   nn   63.

2. filedescr specifying a full description.

    a. filename "nn"

    b. filename$password "nn"

    c. userid/catalog$password "nn"

Filecodes 05, 06, 41, 42, and 43 are implicitly defined for terminal directed I/O and do not need to be mentioned in the RUN command unless I/O is to be directed to a file. Other logical file codes can be terminal-directed by specifying a descriptor of the form filename "nn", where "nn" is the desired filecode.

The I-D-S/II files required for compilation and execution should also be specified in the #fe list. #fe should contain the file descriptor for the 6* subschema file required for compilation with the alternate name "6*".

Example:

    FORTY/DML/6STAR"6*"

#fe should also contain the file descriptors for the I-D-S/II files required for execution that include:

1. Schema File - This file must have the alternate name "1.". If an alternate filecode was specified in the DMCL schema entry, it must have that alternate name.

2. Data Base Area and Key Files - These random files must have alternate names which are the same as the filecodes defined in the DMCL entry.

3. Statistics File - If the STAT option is specified and the output is to be written to a file, the desired file descriptor with the alternate name "P." should be entered in the #fe list.

   Example:

```
FORTY/DML/SCHEMA"L."
FORTY/DML/AREA1"A1"
FORTY/DML/KEY1"K1"
FORTY/DML/STATUS"P."
```

FORTRAN TIME SHARING SYSTEM RUN COMMAND

The FORTRAN time sharing RUN command can be written as either RUN, RUNH, FRN, or FRNH. The RUNH form is used to display a heading line on the terminal giving date and time. Any of the seven following options can be specified with the RUN (or RUNH) command:

```
FRN [H] [-nnn] [fs] [= fh] [; fc] [(opt [,... ])] [ulib]] [#fe]
```

-nnn the maximum processor time (in seconds) the compiled object program is allowed to run during execution.

fs the set of file descriptors (separated by semicolons) for source files in the standard BCD card image format, in compressed card image format (COMDK), or in time sharing ASCII standard system format, and/or descriptors for binary card image object files. These files serve as inputs to the compiler and/or the time sharing loader. Concatenation of source files is provided by using a separate semicolon between each file descriptor.

Where a BCD or COMDK source file is supplied (media code 1 or 2), fs may also include a descriptor for a BCD alter file. The alter file must begin with a $ UPDATE card and must be in alter number sequence. If there are many BCD or COMDK source files in the list, the alter file updates the first source file.

If the FORTRAN program contains I-D-S/II DML statements, fs should also contain the file descriptor for the subschema control structure file. If data base procedures are required and are not supplied as part of the FORTRAN source program, file descriptors for the procedure object files should also be listed here. The list fs can consist of a single file descriptor that points to a previously generated system-loadable (H*) file.

A file descriptor consisting of the single character * indicates the current file (*SRC). The fs list is optional, and when missing indicates that only the current file (*SRC) is to be compiled.

fh a single file descriptor of a random file into which the system loadable file (H*) produced by the general loader is saved if the compilation is successful. This file is written if no fatal errors occur during compilation. If the named file does not exist, a permanent random file of 36 blocks (llinks) is created and added to the user's catalog. If the field is missing, no temporary H* file is created. If this is the case, the time sharing loader creates a complete bound memory-image of the object execution program, "releases" itself via DRL RELMEM, and enters the execution directly.

If the time sharing loader indicates that outstanding SYMREFs exist, any reference to them during object program execution causes abnormal termination via a DRL ABORT.

;fc a single file descriptor (preceded by a semicolon) of a sequential file into which the compiler is to place the binary object (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs.

If the amed file does not exist, a permanent linked file of three blocks (llinks) is created and added to the user's catalog. This file expands as necessary up to a maximum of 20 blocks (llinks), to hold the object deck(s),. When C* is specified, a compiler temporary file (*1 scratch file) of 48 blocks (llinks) is defined and its name is placed into the AFT.

(opt) a list of comma-separated compiler/loader options available in the time sharing based FORTRAN system. Those options available with the YFORTRAN RUN command but not specified here are not currently used with the FORTRAN RUN command. They are ignored if specified (default options are underlined).

BCD - The internal character set for object program execution is BCD. If applicable, this option must be specified whenever the loader is called. This is required for compile, compile and load, and load activities; it is not required for execute only runs (from the H* save file). The user should <u>not</u> load object deck files compiled under different options (<u>i.e.</u>, one under BCD and another under ASCII) since execution results would be unpredictable. The BCD option cannot be specified if the DML option has also been selected.

<u>ASCII</u> - Internal character set for the object program execution is ASCII.

FORM - Source is in "fixed" format (LNO is not valid with FORM).

<u>NFORM</u> - Source is in "free" format.

<u>LNO</u> - Source is line-numbered (default option if FORM is not specified).

NLNO - Source is not line-numbered (default option if FORM is specified).

OPTZ - The object module is optimized.

<u>NOPTZ</u> - The object module is not optimized.

NWARN - No compilation warning messages are printed, although fatal messages are printed.

FDS - Enables the FORTRAN Debugging System (FDS).

DML - The Data Manipulation Language (DML) facility of I-D-S/II is invoked. If DML is specified, the necessary I-D-S/II files must also be specified in the RUN command. The BCD option cannot be used with the DML option.

STAT - The I-D-S/II statistics are printed. If a sequential file with the alternate name "P." exists in the AFT, the I-D-S/II statistics and abort codes are written to that file. The file is written as a BCD file and can be converted to an ASCII file for examination from a terminal by the command "CONV file descriptor". If "P." does not exist in the AFT, the statistics and abort codes are written to the terminal. If the STATS option is not specified, the I-D-S/II statistics are not printed and the fatal abort codes are directed to the terminal. A FINISH statement must be included in the FORTRAN program in order to receive any statistics. This option is valid only if the DML option is specified.

LDEL - Logical record delete is requested. The default is physical record deletion. This option is valid only if the DML option is specified.

The following remaining options concern the loading process:

<u>GO</u> - The program is executed at the successful completion of the compile-load process.

NOGO - The program is not executed at the completion of the compilation. If specified, the object program is loaded and saved. If no object (H*) save file is specified, only the compilation is performed (the General Loader is not called).

ULIB - File descriptors (separated by semicolons) exist following the end of the options field that allocate user libraries to be searched for missing routines prior to searching for them in the system library.

<u>NOLIB</u> - No user libraries are to be used. Specification of user libraries in this case causes a RUN diagnostic.

CORE - nn where nn is additional memory (mod 1024) to be added to the standard time sharing loader allocation of 25K. This should be done if the message " F PROGRAM EXCEEDS STORE SIZE" appears. The compiler attempts to estimate the space requirements for the load process by accumulating the size of the generated memory, .DATA. region, labeled common and blank common for each subprogram compiled; then adding a constant (11K for the standard library) to arrive at the size of a load space requirement. If the message "NOT ENOUGH CORE TO RUN JOB" appears, TSS allocation is too small to compile/load this program.

MAP - A memory map is produced after loading.

ulib - a list of file descriptors (separated by semicolons) pointing to random files containing user libraries to be searched before the system library. This list must be provided by the user when the ULIB option is specified. Up to nine user library files can be specified.

#fe - A list of file descriptors (the first preceded by a number sign) for files required during execution. Each catalog/file description is separated by a semicolon. The file description can be in any of the following formats:

1. filename in the form filename "nn", represents a logical file code referenced by the I/O statements in the program where 01 nn 63.

2. f ledescr specifying a full description.

   "nn"
   filename "nn"
   filename$password "nn"
   userid/catalog$password "nn"

   Filecodes 05, 06, 41, 42, and 43 are implicitly defined for terminal directed I/O and need not be mentioned in the RUN command unless I/O is to be directed to a file. Other logical file codes can be terminal directed by specifying a descriptor of the form "nn", where "nn" is the desired filecode.

   The I-D-S/II files required for compilation and execution should also be specified in the #fe list. #fe should contain the file descriptor for the 6* subschema file required for compilation with the alternate name "6*".

   Example:

   ```
   FORTY/DML/6STAR"6*"
   ```

   #fe should also contain the file descriptors for the I-D-S/II files required for execution that include:

   o Schema File - This file must have the alternate name "1.". If an alternate filecode was specified in the DMCL schema entry, it must have that alternate name.

   o Data Base Area and Key Files - These random files must have alternate names which are the same as the filecodes defined in the DMCL entry.

   o Statistics File - If the STAT option is specified and the output is to be written to a file, the desired file descriptor with the alternate name "P." should be entered in the #fe list.

   Example:

   ```
   FORTY/DML/SCHEMA"L."
   FORTY/DML/AREA1"A1"
   FORTY/DML/KEY1"K1"
   FORTY/DML/STATUS"P."
   ```

Example:

1. Create a random file of 50 llinks, with general read permissions to contain the user's library with the ACCESS subsystem. ACCESS CF,/ULIB1,B/50,50/,R,MODE/R/

2. Listing of a deck setup for creating and saving a user library file (through JRN or batch).

```
1      8        16
$      IDENT    .....
$      USERID   UMC$PASSWD
A$     FILEDIT  NOSOURCE,OBJECT,INITIALIZE
$      FILE     R*,F1S,10L
$      DATA     *C,,COPY
$      SELECTD  UMC/OBJDECK1
$      SELECTD  UMC/OBJDECK2
$      SELECTD  UMC/OBJDECK3
$      ENDEDIT
$      ENDCOPY
A$     PROGRAM  RANLIB
$      PRMFL    A4,W,R,UMC/ULIB1
$      FILE     R*,F1R,10L
$      ENDJOB
```

Alternate Named Files

For files required during execution, the programmer can designate an alternate name by using the following format:

filedescr "altname"

where: altname = nn; attaching the logical file code nn to the specified file.

Examples:

1. RUN#"10"

   If a given file descriptor consists of only a two-digit logical file code not enclosed within quotation marks, a temporary file is created unless a quick-access permanent file with the same name already exists. The PERM command can subsequently be used to make the temporary file permanent. Alternatively, such temporary files can be made permanent at the time the user logs off.

2. RUN PROGRAM#10

   If no file exists in the user's catalog with the name 10, a linked temporary file is created with that name and I/O that was directed to the logical file code 10 is routed to the temporary file.

The fe list of the RUN command serves two additional functions: the creation of a file control block, and the association of the logical file code with some specific file, or the terminal. When this association involves a catalog file descriptor, that file is accessed (or created) and added to the user's available file table (AFT); the file is then allocated to the process. This is analogous to the allocation by the $ PRMFL and $ FILE control cards in a batch operation.

When a file is first referenced by an executing program, a general file "oper" function is invoked. At this time, the file control block becomes involved in one of three ways:

1. There is no file control block for the referenced file.

2. The file control block indicates that the terminal is to be used.

3. The file control block indicates that a file is to be used.

If there is no file control block, one is automatically generated indicating that a file is to be used. When the file control block indicates that the terminal is to be used, the device attachment is completed and I/O proceeds. When the file control block indicates that a file is to be used (cases 1 and 3), the AFT is searched. If a match is found (i.e., an allocated file has a two-digit file code/name equivalent to the file description in the I/O statement), attachment is made to that file and I/O proceeds. If no match is found (i.e., there has been no file allocation for the current file designator), a comment is displayed on the terminal identifying the undefined file designator.

3. FILE XX NOT IN AFT. ACCESS CALLED

where: XX is the two-digit file designator being referenced by the running program.

At this point, the ACCESS subsystem is called (as indicated by the above message) and displays:

FUNCTION?

Commands can now be given to ACCESS. When the dialog is finished, ACCESS returns to the user's program. The "open" routine then makes a fresh search of the AFT. If a match is now found (indicating some file has been accessed), attachment is made to that file and I/O proceeds. If a match is not found, the file control block is changed to indicate attachment to the terminal and I/O proceeds.

Consider that PROGRAM contains I/O statements with a file designator of 10 and the following dialogue transpires:

```
*FORTRAN
*OLD PROGRAM
*RUN

FILE 10 NOT IN AFT. ACCESS CALLED

FUNCTION?
```

If the response is a carriage return, the terminal is used for file 10. If the response is

AF,/MYFILE"10",R,W

the ACCESS subsystem accesses the file MYFILE of the user's master catalog under the alternate name 10 with read and write permissions. ACCESS then repeats the query "FUNCTION?". If the user now responds with a carriage return, I/O for file 10 is directed to MYFILE.

One additional option exists for the purpose of collecting the results of a compiler abort. If at the time the RUN command is issued there exists a file in the AFT of name ABRT, that file is allocated to the compilation activity as file code *F. In the event of a compiler abort, a memory dump and symbolic display of the internal tables is written to this file in a form suitable for printing.

ACCESSING I-D-S/II FILES REQUIRED FOR EXECUTION

The I-D-S/II files necessary for execution can be accessed by listing them in the #fe list of the RUN command as specified above or by the time sharing GET command. Another alternative is to use calls to the supplied FORTRAN subroutine ATTACH.

Example:

```
CALL ATTACH (1,"FORTY/DML/AREA1""A1"";",1,0,ISTAT,)
```

The file is placed in the AFT under the alternate name "A1" which is the file code specified in the schema DMCL. The schema file 1* cannot be accessed in this way because 1* is necessary for the execution of the INVOKE statement, and INVOKE must be the first executable statement.

First Line RUN Command

The RUN command can be designated as the first line or lines of the source program. This is useful when running FORTRAN programs with DML statements because the RUN command may require several lines of input to specify all the I-D-S/II files. The following rules apply to the first line of the RUN command.

1. This feature is available on time sharing ASCII files only.
2. The line can be in the current file (*SRC) or a referenced permanent file; however, it must begin with the first line of the first source file.
3. The first two characters following the line number must be *# with no embedded blanks.
4. Multiple *# lines can appear in a source file, provided the total number of characters does not exceed 480 (six 80-character lines).
5. The lines must conform with the RUN syntax continuation (i.e., each line, except the last, must be terminated by one of the following field-separating delimiters: equal sign, left parenthesis, right parenthesis, semicolon, or number sign).
6. The line(s) are treated as comment line(s) by the FORTRAN compiler.
7. The first line contained RUN command can be overridden by indicating save files, options, or concatenation on the RUN type-in.

Example:

```
*FORTRAN
*NEW
*010*#RUN   *(20,30)=HSTAR(BCD,NOGO)
*020  PRINT, "HELLO DOLLY..."
*030  STOP; END
*RUN (Invokes first line syntax)
```

DML Example:

```
*#RUNH*;FORTY/DML/CSTAR=HSTAR(DML)#FORTY/DML/6STAR"6*";
2*#FORTY/DML/SCHEMA"1.";
3*#FORTY/DML/AREA1"A1";FORTY/DML/KEY1"K1"
```

TSS RUN Examples

1. RUN

   The current *SRC FORTRAN source file is compiled and executed.

2. RUNH-20 FR001=HSTAR; CSTAR1 (ULIB) ABC; XYZ #

   INPUT "01" ; OUTPUT "02"

   FORTRAN program file FR001 is to be compiled and executed. The H* is saved on file HSTAR and C* on file CSTAR1. For the execution, the random user libraries ABC and XYZ are scanned for outstanding SYMREFs in FR001. Logical file codes 01 and 02 have been used as alternate names for the quick-access permanent files INPUT and OUTPUT. A heading line for the date and time is displayed and the object program is limited to 20 seconds of execution time.

3. RUN #"10"

   The current *SRC file is compiled and executed and I/O through logical file code 10 is directed to/from the terminal.

4. RUN BCDIOM = CSTAR2 (BCD,NOGO)

   FORTRAN file BCDIOM is compiled and the object deck is saved on file CSTAR2. The object file is to be executed in BCD mode.

5. RUN HSTAR #02

   Execute a previously bound and saved H* file. The quick-access file "02" is accessed by the RUN subsystem. If no such file exists, a temporary file is created.

6. RUN = HSTAR (TIME=60, CORE=22, ULIB) SEARCH

   Compile and execute the current *SRC file, saving the bound H* file on random file HSTAR. Limit the compile time to 60 seconds and increase the memory limits. The random user library 'SEARCH' is searched to satisfy outstanding SYMREFs prior to searching the standard system library.

7. RUNH *(10,190); SCRLIB(300,)

   Compile and execute the program by concatenating the current file lines 10 through 190 and file SCRLIB lines 300 through the last line of the file.

8. RUN *; CSTAR1; CSTAR2

   Compile and execute the current *SRC file and bind it with two previously saved C* files: CSTAR1 and CSTAR2.

DML TSS Example

```
RUN *;FORTY/DML/CSTAR=(DML,STAT)#FORTY/DML/6STAR"6*";
FORTY/DML/SCHEMA"1.";FORTY/DML/AREA1"A1";
FORTY/DML/KEY1"K1";FORTY/DML/STATUS"P."
```

The current *SRC file is compiled using the subschema file "6*" and bound with the subschema control structure. The resulting object code is executed using the schema file ("1."), one data base area file ("A1"), and one data base key file ("K1"). The I-D-S/II statistics and abort codes are written to the file "P.".

Batch Activity To Build Time Sharing H* File

The following sample program illustrates a method of building a time sharing H* file in batch mode.

```
1      8          16
--------------------------------------------
$      SNUMB      .....
$      IDENT      .....
$      LOWLOAD    100
$      USE        .GRBG./36/
$      OPTION     NOFCB,NOGO,SAVE/object
$      USE        .GTLIT,.TSGF.,.FTSU.,.FXEMA
A$     FORTY      NFORM,NLNO,ASCII
$      SELECTA    source program file
A$     EXECUTE    DUMP
$      PRMFL      H*,W,R,Hstar file
$      ENDJOB
```

Time Sharing System RUNL Command For Link/Overlay

When a bound object program is too large for execution under time sharing, segmentation is achieved by using a special form of the RUN command (RUNL) to link/overlay H* files that are to be constructed. When the RUNL command is used, a PSTR printout can be obtained with the YFORTRAN system but not with the FORTRAN system.

Before the RUNL command can be used, a separate RUN command with the NOGO option must have been specified to create each of the C* files that will be needed in the RUNL command. This command can be written as RUNL or RUNLH where the latter form displays a heading line with the current date and time (and SNUMB if YFORTRAN), with the format

```
RUNL H  [C*file list] [= H*file] [(options)] [ulib files] [; link list]
```

C* file list - The set of file descriptors for the binary object files for the nonoverlayed main program link.

H* file - A single file descriptor of a random file into which the system loadable file produced by the loader is saved if the load process is successful. If the named file does not exist, a file of 216 llinks (random temporary) is created.

(options):

ULIB - File descriptors exist at the end of the options field that locate user libraries to be searched prior to searching the system library. The load process for each link involves searching the same set of user libraries first.

CORE = nn - The YFORTRAN memory requirements are set to nn+9K or 29K, whichever is larger. If not specified, nn is set to 20K.

The FORTRAN link loader memory requirement is nnK if nn < 23K or 23K + nnK if nn > 23.

NAME = name - Provides a name for the main link of the saved H* file; when not provided, the name "//////" is used.

MAP - If the user has previously defined a file with the name PSTR, a load map of the link/overlay save file is written to that file. Otherwise, a temporary file is created by that name and the output is written to that file. This feature is currently available only under the YFORTRAN system.

GO - Allows a user to enter execution directly from the RUNL command (the default is NOGO). The user must provide for run time file definition and dynamic attaching through "CALL ATTACH", etc. If it is necessary to specify through RUN the necessary object time files, the user must explicitly use the RUN command after creating the link/overlay H* file.

Example:

<u>RUN HSTAR#INPUT"01";OUTPUT"02"</u>

link list - A sequence of link phrases wherein each link phrase is used to specify the position at which segmentation is to take place. When the link phrase is encountered in the RUNL command, all object deck files for the link being terminated have been copied to the loader input file R*. The link phrase is parsed, resulting in the generation of a $ LINK card image and possibly a $ ENTRY card image being written to R*.

<u>Formats</u>

LINK(name1 ,name2) C*file list for name1

LINK(name1 ,name2,entry) C*file list for name1

LINK(name1 ,,entry) C*file list for name1

Where: <u>name1</u> (a five- or six-character constant or variable) is a unique identifier for the new link

<u>name2</u>, if present, is the identifier of the previously loaded link to be overlayed. The new link assumes the origin of the old link. All links to be overlayed are written in system loadable format

entry, if specified, is the name of the desired primary or secondary SYMDEF entry point of a subprogram in the current link

Subprograms contained in any other link can always reference subprograms in the main link. Only links that reside in memory at the same time can reference each other. For example, if link B is loaded as an overlay of link A (LINK (B,A)), the subprograms of link B cannot reference subprograms of link A.

NOTES:

1. To ascertain the size required to allocate a permanent H* save file, create a temporary file by means of RUNL. Then use the PERM command to create a permanent file. The size of the permanent file will automatically be chosen just large enough to contain the "used" llinks in the temporary file.

2. Under YFORT, "PSTR" load map generated by the General Loader in batch can be sent to a remote station or central site printer, if it is a permanent file.

   Example:

```
PERM PSTR;PS           Make file permanent if temp used
SCAN PS
FORM? LOAD             Print number of errors
000 ERRORS
EDIT? YES              For multiple-blank suppression
?BATCH
STATION CODE           Reply XX or carriage return

                         XX = remote station code
                         carriage return = central site printer

$ IDENT                Input batch $ IDENT card
```

   Alternatively, a BMC run in batch can print the file.

3. A temporary H* save file cannot be command-loaded; use the LODT command (not LODX). The YFORTRAN or FORTRAN RUN command should be used, since run time files can then be specified.

4. The name of the main link is //////, unless NAME=name is used as an option. The user must specify the name when loading the H* save file.

5. Creating a multiple-line embedded RUNL command is the best way to deal with a long, complex command.

   Example:

```
1*#RUNLH MAIN; SUB1;SUB2=HSTAR (ULIB,MAP)
2*#FY/SDL7LIB,R;
3*#LINK (A)SUB3;SUB4;
4*#LINK (B,A,ENTRY5)SUB5;SUB6;
5*#LINK (C,B)SUB#;SUB8
```

   Observe rules for line termination.

6. After the loader builds the H* save file containing the links, it is necessary to reload these links in the order required to achieve the program function. Reloading is done by means of a time sharing library routine (FTLK) that has two entries, LINK and LLINK. LINK is callable from the FORTRAN source to load a particular link and transfer control to a predesignated entry within that link. This SYMDEF must be specified in the "entry" field of the link phrase. LLINK can be called to load a particular link and return control to the place in the program at which LLINK has been called. The two calls are as follows:

   ```
   CALL LINK ("A     ")
   CALL LLINK ("B     ")
   ```

   The link names must be either five or six characters in length and blank-filled as needed.

7. When using FORTRAN random I/O, the CALL RANSIZ statement must be placed in the main link. This ensures proper file wrapup by forcing the random I/O subroutine FRRD to reside with the main link in memory at all times.

8. The main link in a link/overlay run must contain some input/output when the Hstar file is to be executed in the time sharing mode.

9. The RUNL command cannot be used to process octal patch corrections under the FORT system.

Example Of RUNL Inputs And Link H Creation

Ten subroutines plus a main program are to be executed under time sharing. The first overlay (link A), is to have three subroutines; the second overlay (link B), four subroutines; and the third overlay (link C), three subroutines.

1. Compile and save the C* object deck files (CSTAR) for each program.

   ```
   RUN MAIN =;CSTAR1(NOGO)
   RUN SUBA;SUBB;SUBC =;CSTAR2(NOGO)
   RUN SUBD;SUBE;SUBF;SUBG =;CSTAR3(NOGO)
   RUN SUBH;SUBI;SUBJ =;CSTAR4(NOGO)
   ```

2. Create a link overlay H* file (HSTAR) using RUNL.

   ```
   RUNL CSTAR1 = HSTAR(ULIB,MAP) ULIB1;
   LINK(A) CSTAR2; LINK(B,A,ENTRYB)CSTAR3;LINK(C,B) CSTAR4
   ```

3. Load and execute the H* save file specifying core limits and run-time input/output files.

   ```
   RUN HSTAR=(CORE=35K)#INPUT"41";OUTPUT"13"
   ```

Example Of LINK/LLINK Usage

1. Compile and save the C* object deck files for the main program and the two subroutines.

```
010 PRINT,"MAIN EXECUTING"
020 CALL LLINK ("A     ")
030 CALL SUBA
040 CALL  INK ("B     ")
050 STOP;END
```

RUN =;MAIN(NOGO)
NEW

```
010 SUBROUTINE SUBA
020 PRINT,"LINKA EXECUTING"
030 RETURN; END
```

RUN=;ALINK(NOGO)

```
010 SUBROUTINE SUBB
020 PRINT, "LINKB EXECUTING"
030 RETURN; END
```

RUN=;BLINK(NOGO)

2. Create a link overlay H* file using RUNL.

   RUNL MAIN=HSTAR;LINK(A) ALINK;LINK(B,A,SUBB)BLINK

3. Load and execute the H* file.

   RUN HSTAR
   or
   FRN HSTAR=(CORE=32K)

Example Of Loader Input File

The following control card setup would appear on R* for the example above illustrating the use of LINK/LLINK.

```
 $        LOWLOAD
 $        USE        .GRGB./36/
 $        USE        .GTLIT,.TSGF.,.FTSU.,.FXEMA,.FTLK
 $        OPTION     NOMAP
 $        OPTION     NOGO
 $        OBJECT
 $        DKEND
 $        LINK       A
 $        OBJECT     SUBA
 $        DKEND      SUBA
 $        LINK       B,A
 $        ENTRY      SUBB
 $        OBJECT     SUBB
 $        DKEND      SUBB
A$        EXECUTE
```

Example Of A Time Sharing Session

A comprehensive example of program creation, testing, correction and modification follows. Replies to the user from the system are underlined. Explanations are enclosed in parentheses and are not part of the printout.

```
USER ID - J.P.JONES
PASSWORD--
[redacted]
*FORTRAN
*NEW
*AUTØX - (enter automatic-line-number mode)
*0010     READ,A,B,C
*0020     X1=A*B/C
*0030     X2=A**2;B**2
*0040     ANS=X2/X1
*0050     PRINT 10,X1,X2, ASN#ANS (typing error correction)
*0060 10  FØRMAT(1X,"X1=",F6.S#2,"X2=",F7.2,"ANS=",
*0070     F6.2)
*0080     STØP
*0090     END
*0100     (end automatic mode by carriage return)
*0030     X2=A**2+B**2-C (replacement of line 30)
*SAVE FØRT01
DATA SAVED--FØRT01

*LIST     (display corrected program)
0010      READ,A,B,C
0020      X1=A*B/C
0030      X2=A**2+B**2-C
0040      ANS=X2/X1
0050      PRINT 10,X1,X2, ANS
0060 10   FØRMAT(1X,"X1=",F6.2,"X2=",F7.2,"ANS=",
0070      F6.2)
0080      STØP
0090      END

*RUN    (run program)

= 3.2,10.5,2.2 (type input data)
X1= 15.27X2= 118.29ANS= 7.75    (output - correct,
                                 but poor format)

*0060 10 FØRMAT(1X,"X1="  ,F6.2," X2=",F7.2,"  ANS=",
                            (correct format statement)
*RUN

= 3.2,10.5,2.2
X1= 15.27  X2= 118.29  ANS= 7.75 (improved output format)
*RESAVE FØRT01
DATA SAVED--FØRT01

*BYE (finished)
**RESØURCES USED $  2.08, USED TØ DATE $  263.85= 27%
**TIME SHARING OFF AT 15.421 ON 07/10/79
```

SUPPLYING DIRECT-MODE PROGRAM INPUT

During program execution, keyboard input may need to be supplied to satisfy one or more READ statements in the program. Each time input is required, the equal-sign character, "=", is printed at the terminal. The user begins typing the input immediately following the equal sign.

It is also possible to input data from a paper tape. The actual characters transmitted to the terminal from a READ statement are:

- o carriage return (CR)
- o line feed (LF)
- o equal sign (=)
- o sign-on (X-ON)

The sign-on character activates the paper tape reader if the reader is in the ready state which is achieved by having the paper tape "loaded" and the reader switch set on. Paper tapes which are to be used in this way should end each line with the characters:

- o carriage return (CR)
- o line feed (LF)
- o rubout (RO)
- o sign-off (X-OFF or DC3)

NOTE: The sign-off character, X-OFF, turns off the reader but leaves it in a ready state for any subsequent READs.

Terminal output from the PUNCH statement automatically appends this control information to the end of each line to facilitate the preparation of the tapes. In any event, the user must manually begin such tapes with an appropriate leader of RO characters.

LIMITATIONS IMPOSED BY THE AFT

The AFT allows a maximum of 20 files. This may restrict the running of FORTRAN DML programs in time sharing since a compile-and-execute run requires a source file, subschema files (6* and C*), a schema file (1*), and data base area and key files. If the number of data base areas and key files is large, the run may require more files than allowed in the AFT. Note that the collector file SY** is always present in the AFT.

One way to avoid this difficulty is to use a system-loadable file (H*). The source program can be compiled with the subschema file (6*) and bound with the control file (C*) to produce the H* file. The AFT can then be cleared. The files required for execution can be accessed under their alternate names by the time sharing GET command. Data base area and key files can also be accessed by calls to ATTACH in the FORTRAN source program. The H* can then be run.

Example:

```
RUN DMLTEST;FORTY/DML/CSTAR=HSTAR(DML,NOGO)#FORTY/DML/6STAR"6*"
*REMC
*GET FORTY/DML/SCHEMA"1."
*GET FORTY/DML/AREA1"A1"
*GET FORTY/DML/KEY1"K1"
*RUN HSTAR=(STAT)
```

MEMORY CONSIDERATIONS

Under the FORT or FRN system, the maximum memory allowed for compilation is the initial memory plus a maximum of 75K. The amount of memory available may be limited to less by time sharing itself. If the program is too large to run within these limits, a Y1 (X2) compiler abort occurs. The only way to avoid this situation is to reduce the size of the program.

Under the YFORTRAN system, the maximum memory allowed for compilation is the initial memory plus 3K. If this is not enough memory, the "CORE=" option should be used.

RESTRICTIONS ON LOAD USAGE

It is not possible to ready an area for LOAD in time sharing. The FORTRAN DML statement:

```
READY(ALL REALM=  realm list  ,LOAD)
```

is illegal in time sharing. LOAD usage requires special JCL and must be run in batch. This special JCL is described in Appendix E of the <u>DM-IV (FORTRAN) Program's Reference Manual</u>.

SECTION XII

TIME SHARING ERROR MESSAGES EXPLANATION

Error messages generated by the various time sharing subsystems and by the Time Sharing System Executive program fall into two classes (from the viewpoint of explanations):

- o Error messages that are considered self-explanatory.
- o Error messages that, due to the need for reasonable conciseness in conversational messages, may require further explanation for a given user the first few times that the message is encountered.

All messages falling into the second class are prefixed by a message number, usually enclosed by carets (i.e., <nn>, or in some cases <nn<). Further explanation of these messages is immediately available at the terminal through the HELP subsystem. HELP may be called for either at the subsystem-selection level or at the command level under most major subsystems.

HELP message explanations are listed below, indexed under the associated error message(s). These error messages, in turn, fall into two categories from the viewpoint of origin and applicability.

- o Error messages originating from the time sharing Executive, most of which can be received only by an implementor of a new, not fully debugged, time sharing subsystem during its checkout. These messages are numbered 1 through 49.
- o Error messages originating from the various time sharing subsystems, which would be received by a user of the system. These messages indicate faulty system usage or system malfunction, and are numbered beginning with 50.

  NOTE: On some types of terminals, the carets enclosing the error message number are reproduced as parentheses.

In the following descriptions, generated error messages and their associated HELP subsystem error message explanations are listed by message numbers.

001 - INCORRECT PRIMITIVE

AN ILLEGAL PRIMITIVE HAS OCCURRED IN A COMMAND LIST. CHECK THE COMMAND LIST POINTER IN THE PROGRAM DESCRIPTOR AND THE COMMAND LIST FORMAT AND PRIMITIVES.

002 - location INVALID FILE I/O COMMAND

IN THE CALLING SEQUENCE OF A DRL FOR DISK I/O, THE SEEK, READ or WRITE COMMAND IS INCORRECT. CHECK THE SUBSYSTEM CODE.

003 - location INVALID DCW

IN THE CALLING SEQUENCE OF A DRL FOR DISK I/O, A DCW IS INCORRECT. CHECK THE SUBSYSTEM CODE.

004 - location INVALID DRL ARGUMENT

THE ADDRESS OF A DRL ARGUMENT IS OUTSIDE THE RANGE OF THE PROGRAM. THE NUMBER GIVEN IN THE COMMENT IS THE RETURN FROM THE DERAIL. CHECK THE SUBSYSTEM CODE FOR IMPROPER INITIALIZATION.

005 - BAD DRL CODE

THE ADDRESS OF A DRL CODE IS OUT OF THE RANGE OF USABLE CODES OR ILLEGAL FOR THIS SUBSYSTEM. CHECK THE SUBSYSTEM CODE.

006 - LEVEL OF CONTROL TOO DEEP

THE MAXIMUM NUMBER OF CALLS IN THE PROGRAM STACK OR THE CALLSS STACK HAS BEEN EXCEEDED. IN THE CASE OF THE PROGRAM STACK, THIS MEANS THAT THE SELECTED SYSTEMS PRIMITIVE LIST CONTAINED A CALLP, AND IN TURN, THAT SUBSYSTEMS PRIMITIVE LIST CONTAINED A CALLP, ETC. UNTIL THE LENGTH OF THE PROGRAM STACK WAS EXCEEDED. LIKEWISE, IN THE CASE OF THE CALLSS STACK OF SUBSYSTEMS CALLING OTHER SUBSYSTEMS BY MEANS OF THE DRL CALLSS, THE TABLE LIMIT WAS EXCEEDED. REVIEW THE SUBSYSTEM AND DEPTH OF CALLS.

007 - BAD PROG. DESCRIPTION

IN THE PROGRAM DESCRIPTOR, THE POINTER TO THE COMMAND LIST IS ZERO OR POINTS TO NON-COMMAND LANGUAGE DATA. CHECK THE PROGRAM DESCRIPTOR AND COMMAND LANGUAGE LIST.

008 - LOOP IN PRIMITIVES

A NUMBER OF THE PRIMITIVES ARE EXECUTED ENTIRELY WITHIN THE TSS SCAN MODULE. A COUNTER IS INITIALIZED AT THE ENTRY TO SCAN AND A COUNT KEPT OF PRIMITIVES EXECUTED. WHEN THE COUNT EXCEEDS A GIVEN MAXIMUM, IT BECOMES OBVIOUS THERE IS A LOOP. CHECK THE SEQUENCE OF THE PRIMITIVES FOR THE SUBSYSTEM.

009 - SYSTEM UNKNOWN

THE REQUESTED SUBSYSTEM IS UNKNOWN TO TSS OR IS NOT INCLUDED IN THE SYSTEM FOR THIS INSTALLATION. CHECK THE NAME FOR SPELLING TOO.

010 - PROGRAM TOO LARGE TO SWAP

A SUBSYSTEM IS SO LARGE THAT THE NUMBER OF DCW'S REQUIRED TO LOAD OR SWAP THE PROGRAM EXCEED THE MAXIMUM NUMBER OF DCW'S WHICH CAN BE BUILT. CHECK THE SIZE OF THE SUBSYSTEM. PERHAPS THE SUBSYSTEM EXPANDS ITS CORE LIMITS WITH A DRL ADDMEM. CHECK ALL DRL ADDMEM REQUESTS. SEE LADCW DEFINED IN COMMUNICATION REGION FOR MAXIMUM NUMBER OF DCW'S ALLOWED.

011 - location INCORRECT CORE FILE USAGE

A REQUEST TO MOVE CORE FILE SPECIFIES MORE THAN TEN WORDS TO BE MOVED. CHECK ALL DRL CORFIL REQUESTS.

012 - location PRIVILEGED I/O ATTEMPTED

PRIVILEGED DISK I/O IS RESERVED FOR SUBSYSTEMS WHICH SPECIFICALLY REQUIRE INFORMATION FROM FILES ALLOCATED TO THE TIME SHARING SYSTEM. PLEASE REVIEW THE NEED FOR PRIVILEGED DISK I/O AND JUSTIFY IT WITH THE COMPUTING CENTER.

013 - location USERID NOT PERMITTED

THE DRL USER ID CAN BE USED ONLY BY THE LOGON SUBSYSTEM. CHECK THE SUBSYSTEM CODE.

014 - NOT CURRENTLY ASSIGNED.

015 - location CANNOT RESET USERID

THE LOGON SUBSYSTEM IS EXECUTING A DRL USER ID, BUT THE ID OF THE SPECIFIED U.S.T. IS NON-ZERO. A TERMINATE MUST BE EXECUTED FOR THAT USER BEFORE THE U.S.T. CAN BE REUSED. TRY TO DETERMINE WHY THE TERMINATE WAS BYPASSED, OR WHY NEW SYSTEM WAS SELECTED AFTER LOGON.

016 - location OVERFLOW FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED AN OVERFLOW CONDITION AT THE DESIGNATED LOCATION AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR. THE LOCATION IS RELATIVE TO ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM. THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM. REVIEW YOUR PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

017 - location ILLEGAL OP CODE

THE SUBSYSTEM IN EXECUTION ENCOUNTERED AN ILLEGAL (OR ZERO) OP CODE OR A MME OPERATION AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW YOUR PROGRAM CODE AND INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

018 - Location MEMORY FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A MEMORY FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

019 - Location FAULT TAG FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A FAULT TAG FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

020 - Location DIVIDE CHECK FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A DIVIDE CHECK FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW YOUR PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

021 - (nnnnnn) BAD STATUS SWAP OUT #S

A BAD I/O STATUS HAS BEEN RECEIVED ON A WRITE DRUM FILE #S, THE SWAP FILE. TRY AGAIN. IF PROBLEM PERSISTS, ALERT OPERATIONS. THE PARENTHESIZED NUMBER IS THE STATUS CODE.

022 - (nnnnnn) BAD STATUS SWAP IN #S

A BAD I/O STATUS HAS BEEN RECEIVED ON A READ DRUM FILE #S, THE SWAP FILE. TRY AGAIN. IF PROBLEM PERSISTS, ALERT OPERATIONS. THE PARENTHESIZED NUMBER IS THE STATUS CODE.

023 - (nnnnnn) BAD STATUS LOAD #P

A BAD I/O STATUS HAS BEEN RECEIVED ON A READ DRUM FILE #P, THE TSS FILE. TRY AGAIN. IF PROBLEM PERSISTS, ALERT OPERATIONS. THE PARENTHESIZED NUMBER IS THE STATUS CODE.

024 - location MME FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A MME FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

025 - location LOCKUP FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A LOCKUP FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

026 - location OP-NOT-COMPLETE FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED AN OP-NOT-COMPLETE FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

027 - location COMMAND FAULT

THE SUBSYSTEM IN EXECUTION ENCOUNTERED A COMMAND FAULT AT THE DESIGNATED LOCATION, AND THE SUBSYSTEM DID NOT SPECIFY A FAULT VECTOR.

THE LOCATION IS RELATIVE TO SUBSYSTEM ZERO (SEE EDIT MAP) UNLESS IT IS A MASTER SUBSYSTEM, THEN THE LOCATION IS RELATIVE TO TSS ZERO AND ONE MUST DETERMINE THE LOAD ADDRESS OF THE SUBSYSTEM TO DETERMINE THE FAULT LOCATION IN THE MASTER SUBSYSTEM.

REVIEW THE PROGRAM CODE AND INITIALIZATION OF ADDRESS OR INDEX REGISTERS AS WELL AS THE PROGRAM INPUT FOR INCORRECT DATA BEFORE REQUESTING HELP FROM THE COMPUTING CENTER.

028 - location REWIND ATTEMPTED FOR RANDOM FILE - FILENAME

A RANDOM FILE CANNOT BE SPACED IN THIS MANNER. USAGE OF THE RANDOM FILE IN THE CORRECT MANNER WILL CLEAR UP THE PROBLEM.

029 - ILLEGAL SYSTEM SELECTION

SOME SYSTEMS, NAMELY THE MASTER SUBSYSTEMS, HAVE RESTRICTED THEIR AVAILABILITY TO CERTAIN USERS. YOU DO NOT HAVE PERMISSION TO USE THE SELECTED SUBSYSTEM. SELECT ANOTHER.

30-49 - NOT CURRENTLY ASSIGNED.

<50> FILE filename -- reason text

<50> FILE filename -- reason text

(The two messages above refer to permanent files.)

<50> CURRENT FILE -- reason text

<50< COLLECTOR FILE -- reason text

(The two messages above refer to the temporary files *SRC and SY**, respectively.)

<50> WORK FILE -- reason-text

(The message above refers to all other temporary files.)

ERROR-MESSAGE 50 EXPLANATION: FILE-SYSTEM ERRORS.

THIS MESSAGE IS ISSUED FOR EITHER ONE OF TWO CASES: (1) THE NAMED PERMANENT FILE COULD NOT BE ACCESSED-- <50>, OR COULD NOT BE CREATED--<50< OR (2) A REQUIRED TEMPORARY FILE COULD NOT BE OBTAINED OR EXPANDED. REPLY TO THE QUESTION "GROUP?" AS FOLLOWS FOR A FURTHER EXPLANATION: IF YOUR MESSAGE STATES "NO PERMISSION, NONEXISTENT FILE" OR "INVALID PASSWORD," REPLY "1". IF "FILE BUSY, NO FILE SPACE" OR "ILLEGAL CHAR.," REPLY "2". IF "I/O ERROR, FILE TABLE FULL, DUPLICATE NAME" OR "SYSTEM LOADED," REPLY "3". IF IT STATES "STATUS NN" REPLY "4".

STATUS 01: THE SPECIFIED USER'S-MASTER-CATALOG DOES NOT EXIST. CHECK USER-ID.

STATUS 02: I/O ERROR. THE FILE SYSTEM HAS ENCOUNTERED AN UNRECOVERABLE INTERNAL I/O ERROR. (THIS DOES NOT IMPLY AN ERROR ON YOUR FILE SPACE.) REPORT THE STATUS TO THE CENTRAL COMPUTER SITE. ALSO RETRY.

STATUS 03: PERMISSION DENIED. THE NAMED FILE COULD NOT BE ACCESSED BECAUSE YOU HAVE NOT BEEN ALLOWED THE PERMISSION(S) REQUESTED. IF THE FILE IS ALREADY OPEN, THE PERMISSIONS REQUESTED DO NOT MATCH THE PERMISSIONS WITH WHICH THE FILE IS ALREADY OPENED. THIS STATUS IS ALSO RETURNED BY THE FILE SYSTEM WHEN AN ATTEMPT IS MADE TO OPEN A "NULL" FILE WITH "READ" PERMISSION ONLY.

STATUS 04: FILE BUSY. ANOTHER USER HAS ALREADY ACCESSED THIS FILE WITH AN ACCESS-MODE PERMISSION THAT LOGICALLY EXCLUDES YOUR REQUESTED PERMISSION; I.E., A GRANTED WRITE PERMISSION EXCLUDES ANY OTHER CONCURRENT ACCESSES AND A GRANTED READ PERMISSION EXCLUDES ANY OTHER ACCESS WITH WRITE PERMISSION. THE FILE, THEREFORE, IS TEMPORARILY BUSY TO SOME OR ALL OTHER USERS. (MULTIPLE CONCURRENT ACCESSES OF A FILE WITH READ PERMISSION, ONLY, IS ALLOWED.)

STATUS 05: NONEXISTENT FILE. EITHER THE NAMED FILE DOES NOT EXIST, AT THE CATALOG LEVEL IMPLIED OR SPECIFIED, OR ONE OR MORE NAMES IN THE CATALOG/FILE DESCRIPTION WAS INCORRECTLY GIVEN. CHECK ALL CATALOG/FILE NAMES. THE COMMAND CATALOG MAY BE USED TO LIST ALL OF YOUR CATALOG AND FILE NAMES.

STATUS 06: THE FILE SYSTEM HAS EXHAUSTED ITS SPACE FOR NEW CATALOGS AND FILE DESCRIPTORS. REPORT THE STATUS TO THE CENTRAL COMPUTER SITE, AND TRY AGAIN LATER.

STATUS 07: DEVICE TYPE UNDEFINED. THE DEVICE TYPE THAT YOU SPECIFIED FOR YOUR FILE IS UNDEFINED TO THE SYSTEM.

STATUS 10: THE SYSTEM HAS TEMPORARILY EXHAUSTED THE AVAILABLE FILE SPACE. TRY AGAIN LATER. (ALSO, PURGE ANY UNNEEDED FILES.)

STATUS 11: NON-UNIQUE NAME. THE NEW NAME THAT YOU HAVE SPECIFIED FOR THE CATALOG OR FILE TO BE MODIFIED IS A DUPLICATE OF A CATALOG OR FILE NAME EXISTING AT THE SAME LEVEL.

STATUS 12: MAX. SIZE ERROR. THE NEW MAXIMUM-SIZE SPECIFIED FOR THE FILE TO BE MODIFIED IS LESS THAN ITS CURRENT SIZE. (MAXIMUM SIZE UNCHANGED.)

STATUS 13: NO FILE SPACE. YOU HAVE USED UP ALL THE PHYSICAL SPACE ALLOTTED TO YOU FOR THE CREATION OF FILES. YOU MUST EITHER PURGE ONE OR MORE UNNEEDED FILES, OR OBTAIN A LARGER FILE-SPACE ALLOCATION.

STATUS 14: INVALID PASSWORD. A REQUIRED PASSWORD EITHER HAS BEEN GIVEN INCORRECTLY OR NOT AT ALL. THE GENERAL FORM FOR SUPPLYING PASSWORDS IN A CATALOG/FILE DESCRIPTION IS: NAME$PASSWORD E.G.: /CAT1$ABC/FIL1$XYZ.

STATUS 15: FILE IS ABORT LOCKED.

STATUS 16: FILE WRITE IN BATCH ONLY.

STATUS 17: SEEK ERROR.

STATUS 20: FAILURE IN NAME SCAN.

STATUS 21: UNDEFINED DEVICE.

STATUS 22: DEVICE LINK TABLE CHECKSUM ERROR.

STATUS 23: INCONSISTENT FSW BLOCK COUNT.

STATUS 24: INTERNAL LINK TABLE CHECKSUM ERROR.

STATUS 25: REQUESTED ENTRY NOT ON LINE.

STATUS 26: NON-STRUCTURED FILE ENTRY.

STATUS 27: FILE IN DEFECTIVE STATUS.

STATUS 30: ILLEGAL PACK TYPE.

STATUS 31: ACCESS GRANTED TO IDS FILE.

STATUS 32: COLLECTION FILE ERROR.

STATUS 33: CATALOG/FILE SECURITY LOCKED

STATUS 34: ILLEGAL CHAR. YOU HAVE GIVEN A CATALOG OR FILE NAME, OR A PASSWORD, CONTAINING A CHARACTER OTHER THAN AN ALPHANUMERIC, PERIOD, OR A DASH, WHICH ARE THE ONLY LEGAL CHARS. FOR IDENTIFIERS.

STATUS 35: PERMISSION NOT GRANTED TO LIST OR PURGE REQUESTED CATALOG.

STATUS 36: FILE TABLE FULL. THE NAMED FILE CANNOT BE ACCESSED BECAUSE YOU PRESENTLY HAVE TOO MANY FILES ALREADY ACCESSED (I.E., OPENED). YOU MUST DEACCESS ONE OR MORE OF THESE OPENED FILES. USE THE COMMANDS STATUS FILES, AND REMOVE.

STATUS 37: DUPLICATE NAME. THE FILE NAME SHOWN DUPLICATES A NAME ALREADY IN YOUR AVAILABLE-FILE-TABLE, I.E., AN ALREADY ACCESSED FILE. IF APPROPRIATE, ASSIGN AN ALTERNATE NAME.

STATUS 40: SYSTEM LOADED. THE SYSTEM IS CURRENTLY AT PEAK CAPACITY IN SOME RESPECT, E.G.: CERTAIN INTERNAL TABLE SPACE EXHAUSTED, ETC.

STATUS 41: NO PROTECTION TABLE SPACE AVAIL.

STATUS 42: INVALID FILE CODE OR PAT POINTER.

STATUS 43: INVALID CATALOG BLOCK ADDRESS.

STATUS 44: PERMISSION DENIED - SHARED FILE.

STATUS 45: INVALID SPACE IDENTIFIER.

STATUS 46: CATALOGS BUSY.

STATUS 47 AND 50: SYSTEM MALFUNCTION. REPORT THE STATUS TO THE CENTRAL COMPUTER SITE, AND RETRY.

STATUS 51: CHECKSUM ERROR ON DEVICE.

STATUS 52: DEVICE RELEASED.

STATUS 53: NOT CURRENTLY ASSIGNED.

STATUS 54: NOT CURRENTLY ASSIGNED.

STATUS 55: NOT CURRENTLY ASSIGNED.

STATUS 56: SECURITY PARAMETER - REQUIRED.

STATUS 57: SECURITY PARAMETER - INVALID.

STATUS 60: SITE USED STATUS.

STATUS 61: $FSYS HAS BEEN ENABLED.

STATUS 62: ILLEGAL SUBFUNCTION CODE.

STATUS 63: FILE NOT BEING MONITORED.

STATUS 64: DEADLOCK ON PAGE REQUEST.

STATUS 65: PAGE CURRENTLY BUSY.

STATUS 66: FILE NOT DUPLICATED.

STATUS 67: TDS MONITOR ALLOC ERROR.

STATUS 70: ILLEGAL CHECKPOINT REQUEST.

STATUS 71: ILLEGAL DCW SPECIFIED.

STATUS 72: IMPROPER PROTECTION OPTION.

STATUS 73: INVALID ARGLIST PARAMETER NUMBER.

STATUS 74: SYSTEM JOURNAL NOT CONFIGURED.

STATUS 75: FILE RESTORE LOCKED.

STATUS 76: FILE TDS LOCKED.

STATUS 77: ERR TDS SUBSET PAGES RELEASE.

<51> FILE filename -- I/O STATUS yy

<51< FILE filename -- I/O STATUS yy

(The two messages above refer to permanent files.)

<51> CURRENT FILE -- I/O STATUS yy

<51< CURRENT FILE -- I/O STATUS yy

(The two messages above refer to the *SRC file.)

<51> COLLECTOR FILE -- I/O STATUS yy

<51< COLLECTOR FILE -- I/O STATUS yy

(The two messages above refer to the SY** file.)

<51> WORK FILE -- I/O STATUS yy

<51< WORK FILE -- I/O STATUS yy

(The two messages above refer to all other temporary files.)

where yy is the major hardware status returned by IOS. These status codes are described in the General Comprehensive Operating Supervisor reference manual.

ERROR-MESSAGE 51 EXPLANATION: INPUT/OUTPUT ERRORS

AN UNRECOVERABLE READ OR WRITE ERROR HAS OCCURRED ON THE SPECIFIED FILE. AN ERROR IN READING IS INDICATED BY THE MESSAGE NUMBER GIVEN AS <51>; AN ERROR IN WRITING AS <51<. REPORT THE I/O STATUS NUMBER AND THE READ OR WRITE INDICATION TO THE CENTRAL COMPUTER SITE. ALSO, IN THE CASE OF "CURRENT FILE" or "WORK FILE", LOG OFF AND TRY AGAIN.

<52> CURRENT FILE NOT DEFINED

ERROR-MESSAGE 52 EXPLANATION

THERE IS NO CURRENT (*SRC) FILE DEFINED IN YOUR FILE TABLE. THIS INDICATES EITHER A SYSTEM MALFUNCTION, OR THAT YOU ARRIVED AT THE PRESENT SUBSYSTEM VIA AN ABNORMAL PATH. SUGGEST YOU RESELECT YOUR DESIRED SUBSYSTEM, OR LOG OFF AND RETRY FROM SCRATCH.

<53> LINES IGNORED BY EDIT
....line(s)......

ERROR-MESSAGE 53 EXPLANATION

THE LINE(S) SHOWN WERE NOT MERGED INTO YOUR CURRENT FILE BECAUSE THEY LACKED LINE NUMBERS.

<54> SYSTEM MALFUNCTION--CURRENT FILE ERROR

ERROR-MESSAGE 54 EXPLANATION

THE FORMAT OF YOUR CURRENT FILE WAS FOUND TO BE IN ERROR. REPORT CIRCUMSTANCES TO THE CENTRAL COMPUTER SITE. SUGGEST THAT YOU LOG OFF AND RETRY.

<55> CURRENT FILE TOO LARGE

ERROR-MESSAGE 55 EXPLANATION

THE COMBINED SIZE OF YOUR SOURCE FILE AND MOST RECENT MODIFICATION- OR ADDITION-INPUT IS TOO LARGE TO BE PROCESSED. SUGGEST THAT YOU SPLIT THE TEXT INTO TWO OR MORE FILES, WHICH CAN LATER BE ADJOINED.

<56> NOT CURRENTLY ASSIGNED

057 - RESTRICTED SUBSYSTEM

THE CENTRAL COMPUTER SITE HAS RESTRICTED THE USE OF THIS SYSTEM. THIS MAY BE A TEMPORARY RESTRICTION BECAUSE OF CURRENT LOAD OR A PERMANENT RESTRICTION. PLEASE NOTIFY THE CENTRAL COMPUTER SITE FOR FURTHER DETAILS.

<58> ENTRY LOC < 100

ERROR-MESSAGE 58 EXPLANATION

THE SUBSYSTEM PROGRAM TO BE EXECUTED DOES NOT HAVE THE INITIAL 100-WORD DATA AREA THAT IS REQUIRED OF TSS SUBSYSTEM PROGRAMS.

<59> FILE filename NOT IN TSS FORMAT

ERROR-MESSAGE 59 EXPLANATION

A FORMAT ERROR WAS DETECTED ON THE NAMED FILE. EITHER THE FILE IS NOT A TSS-GENERATED FILE, OR A SYSTEM MALFUNCTION HAS OCCURRED. IN THE LATTER CASE, REPORT THE CIRCUMSTANCES TO THE CENTRAL COMPUTER SITE, AND RETRY THE COMMAND.

<60> NO DATA ON FIL filename

ERROR-MESSAGE 60 EXPLANATION

THE REQUESTED FILE CONTAINS NO USER'S DATA; THE IMPLICATION IS THAT NO DATA HAS BEEN SAVED ON THIS FILE SINCE ITS CREATION.

061-063 - NOT CURRENTLY ASSIGNED.

<064> - EXECUTE TIME LIMIT EXCEEDED

THE PROGRAM TIME LIMIT SPECIFIED BY THE USER AND/OR THE INSTALLATION HAS BEEN EXCEEDED BY THE OBJECT PROGRAM.

<065> - OBJECT PROGRAM SIZE LIMIT EXCEEDED

THE SIZE OF THE OBJECT PROGRAM HAS EXCEEDED THE INSTALLATION SPECIFIED LIMIT.

<066> - SPAWN UNSUCCESSFUL -- STATUS N

A SUBSYSTEM WAS UNABLE TO SPAWN A JOB TO BATCH FOR COMPILATION AND/OR LOADING. THE REASON CODE "N" DESCRIBES ONE OF THE FOLLOWING:

1 - UNDEFINED FILE (FILE NOT IN AFT)
2 - NO SNUMB
3 - DUPLICATE SNUMB
4 - NO PROGRAM NUMBER AVAILABLE
5 - ACTIVITY NAME UNDEFINED
6 - ILLEGAL USER LIMITS (TIME, SIZE, ETC.)
7 - BAD STATUS (R/W J*)
8 - NO FILE SPACE AVAILABLE FOR PUSH-DOWN FILE
9 - NO *J FILE PROVIDED

<067> - (Error message text)

THE ERROR DESCRIBED IN THE MESSAGE HAS BEEN DETECTED BY TSS WHILE IN COMMAND FILE OR DEFERRED PROCESSING MODE. THE MODE HAS BEEN DISCONTINUED DUE TO ITS OCCURRENCE.

068 - NOT CURRENTLY ASSIGNED.

069 - ERROR-MESSAGE 69 EXPLANATION

THE ERROR DESCRIBED IN THE MESSAGE HAS BEEN DETECTED BY TEX WHILE IN FILE EXECUTION MODE.

070-133 - NOT CURRENTLY ASSIGNED.

134 - location INVALID DRL FILACT FUNCTION

THE DRL FILACT CALLING SEQUENCE CONTAINS AN INVALID FUNCTION NUMBER.

135 - <u>location</u> PRIVILEGED DRL FILACT REQUEST

USER CANNOT ACCESS THE SYSTEM MASTER CATALOG.

136 - NOT CURRENTLY ASSIGNED.

137 - NOT CURRENTLY ASSIGNED.

138 - <u>location</u> NO TAP* FILE FOR DRL TAPEIN

THE TAP* FILE IS UNDEFINED FOR TAPE INPUT.

139 - ERROR IN WRITING TAP* FILE

AN ERROR OCCURRED WHILE WRITING IN THE TAP* FILE.

APPENDIX B

EXECUTIVE ERROR MESSAGES

Error Code	Text
1	001-INCORRECT PRIMITIVE
2	002-(dddddd)INVALID FILE I/O COMMAND
3	003-(dddddd)INVALID DCW
4	004-(dddddd)INVALID DRL ARGUMENT
5	005-(dddddd)INVALID DRL CODE
6	006-LEVEL OF CONTROL TOO DEEP
7	007-BAD PROG. DESC.
8	008-LOOP IN PRIMITIVES
9	009-SYSTEM UNKNOWN
10	010-PROGRAM TOO LARGE TO SWAP
11	011-(dddddd)INCORRECT CORE FILE USAGE
12	012-(dddddd)PRIVILEGED I/O ATTEMPTED
13	013-(dddddd)DRL USERID NOT PERMITTED
14	(dddddd)ILLEGAL DRL RELMEM REQUEST
15	015-(dddddd)CANNOT RESET USER ID
16	016-(aaaaaa)OVERFLOW FAULT
17	017-(aaaaaa)ILLEGAL OP CODE
18	018-(aaaaaa)MEMORY FAULT
19	019-(aaaaaa)FAULT TAG FAULT
20	020-(aaaaaa)DIVIDE CHECK FAULT
21	021-(ssssss)BAD STATUS - SWAP OUT
22	022-(ssssss)BAD STATUS - SWAP IN
23	023-(ssssss)BAD STATUS - LOAD
24	(dddddd)TALK PERMISSION NOT GRANTED
25	(dddddd)WRITE ATTEMPTED ON READ-ONLY FILE - ffffffff
26	(dddddd)READ ATTEMPTED ON EXECUTE-ONLY FILE - ffffffff
27	024-(aaaaaa)MME FAULT
28	028-(dddddd)REWIND ATTEMPTED FOR RANDOM FILE - ffffffff
29	029-ILLEGAL SYSTEM SELECTION
30	134-(dddddd)INVALID DRL FILACT FUNCTION #
31	135-(dddddd)PRIVILEGED DRL FILACT REQUEST
32	138-(dddddd)NO TAP* FILE FOR DRL TAPEIN
33	139-ERROR IN WRITING TAP* FILE
34	(dddddd)DRL ABORT - CANNOT WRITE ABRT FILE
35	(dddddd)DRL ABORT - ABRT FILE WRITTEN
36	NOT ENOUGH CORE TO RUN JOB
37	SORRY-OUT OF SWAP SPACE. TRY AGAIN.
38	(dddddd)FILE ADDRESS ERROR
39	(dddddd)DRL ABORT - ABRT FILE I/O ERROR
40	(dddddd)DRL ABORT - ABRT FILE TOO SMALL
41	(ssssss)BAD STATUS FOR DRL SAVE/RESTOR - ffffffff
42	(dddddd)H* FILE NOT IN AFT - ffffffff
43	064-EXECUTE TIME LIMIT EXCEEDED
44	025-(aaaaaa)LOCKUP FAULT
45	065-OBJECT PROGRAM SIZE LIMIT EXCEEDED
46	(dddddd)INCORRECT ENTRY TO DRL TASK
47	(dddddd)H* PROGRAM NAME UNDEFINED - ffffffff
48	(dddddd)H* FILE CATALOG FULL - ffffffff
49	(dddddd)TALLY OR CHARACTER COUNT INCORRECT
50	(dddddd)BAD DRL SAVE DATA LOC
51	(dddddd)H* FILE NOT INITIALIZED - ffffffff

52 (dddddd)H* FILE MUST BE RANDOM - ffffffff
53 026-(aaaaaa)OP-NOT-COMPLETE FAULT
54 (dddddd)H* FILE PROGRAM NAME REQUIRED - ffffffff
55 027-(aaaaaa)COMMAND FAULT
56 (dddddd)LINKED FILE I/O CANNOT SPAN 63 LLINKS - ffffffff
57 UNASSIGNED
58 (dddddd)INVALID TIME FOR DRL GWAKE
59 UNASSIGNED
60 (dddddd)INVALID SNUMB FOR DRL JOUT
61 (dddddd)PRIVILEGED DRL
62 (dddddd)INVALID DRL JOUT FUNCTION #
63 MEMORY PARITY ERROR
64 SY** I/O ERROR

legend:

ffffffff - Name of the file associated with the error.
dddddd - Location of the derail which caused the error.
aaaaaa - Address in the subsystem at which the error occurred.
ssssss - Bad file I/O status received.

APPENDIX D

OCTAL-ASCII CONVERSION EQUIVALENTS

OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.
000	NULL	040	b̸	100		140	GRA
001	SOH	041	EXP	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046		106	F	146	f
007	BELL	047	'	107	G	147	g
010	BSP	050	(	110	H	150	h
011	HT	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FFD	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	LBK	173	LBR
034	FS	074	LTN	134	RSL	174	VTL
035	GS	075	=	135	RBK	175	RBR
036	RS	076	GTN	136	CFX	176	TLD
037	US	077	?	137		177	DEL

DEFINITIONS

Communications Control

ACK	Acknowledgment
CAN	Cancel
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
DLE	Data Link Escape
EM	End of Medium
ENQ	Enquiry
EOT	End of Transmission
ESC	Escape (Alternate Mode)
ETB	End of Transmission Block
ETX	End of Text
NAK	Negative Acknowledgment
SOH	Start of Heading
STX	Start of Text
SUB	Substitute Character
SYN	Synchronous Idle

Form Effectors

BSP	Backspace
CR	Carriage Return
FFD	Form Feed
HT	Horizontal Tabulation
LF	Line Feed
VT	Vertical Tabulation

Item Separators

FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator

Text Material

BELL	Bell, or other attention signal
CFX	
DEL	Delete (Rubout)
EXP	'.
GRA	
GTN	
LBK	
LBR	
LTN	
NULL	Null
RBK	
RBR	
RSL	
SI	Shift In
SO	Shift Out
SP	Space
TLD	
VTL	Vertical Line

INDEX

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 66)/6000 TIME SHARING SYSTEM REFERENCE MANUAL	ORDER NO.	DJ31-00
		DATED	OCTOBER 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell